yorick Documentation

Release 0.1pre

Adam Brenecki

March 11, 2014

Whenever you start a programming project (or a book, or anything you do that involves files on a computer), you often end up doing the same initial steps, without much variation.

Yorick allows you to automate this by creating "skeletons" - templates that you can "construct" to create a boilerplate project. In the process of constructing, a skeleton can prompt the user for variables (project name, for instance), and have those variables substituted appropriately into the skeleton.

A collection of skeletons for different types of projects is called a "closet". Yorick automatically gives you a default closet to keep all of your own skeletons in, and the ability to add other people's closets alongside it. You can keep your closet to yourself or open it up to the world on GitHub (a bit like dotfiles).

Contents:

CHAPTER 1

Constructing a Yorick skeleton

Projects are constructed from a skeleton using the *yorick construct* command.

```
$ yorick construct eggs
Enter a name for your project.
project_name> spam
Constructing... Done.

$ find .
./spam/
./spam/__init__.py
./README.md

$ cat README.md
# spam

Insert a readme for spam here.
```

Instead of being prompted for variables interactively, you can specify them on the command line.

```
$ yorick construct eggs project_name=spam
Constructing... Done.
```

Skeletons are stored inside 'closets'. All of your closets are in the `~/.yorick/` folder.

By default, skeletons are taken from the 'default closet', which is located at `~/.yorick/__default__/`. (So, the *eggs* skeleton would be at `~/.yorick/__default__/eggs/`.)

To construct a skeleton from a different closet, use the form *yorick construct* <*closet*>.<*skeleton-name*>. For instance, the command `yorick construct fred.spam` would construct the skeleton located at `~/.yorick/fred/spam/`.

To construct a skeleton that is somewhere other than one of the closets in your *.yorick* directory, use the *-p* flag to specify a path to it, for example *yorick construct -p ~/blah/myskeleton*.

CHAPTER 2

Writing your first Yorick skeleton

Use the `yorick create-skeleton` command to create a new skeleton. In this example, we'll create a skeleton called `eggs`.

```
$ yorick create-skeleton
Enter a name for your skeleton.
skeleton_name> eggs
Constructing... Done.
You can now edit your skeleton at ~/.yorick/__default__/eggs/
```

Go ahead and open that directory in your favourite text editor.

2.1 Writing the Configuration File

Open the file `-yorick-meta/config.yml`.

2.2 Adding a File

CHAPTER 3

Yorick skeleton reference

3.1 Skeleton Properties

These are the properties that go in `-yorick-meta/config.yml`.

- `id` - A UUID for the skeleton. This should be generated for you by `yorick create-skeleton`, and you shouldn't change it unless you're creating a new skeleton by copying a different one.
- `description` - A description for the skeleton.
- `variables` - A dictionary of variables, as specified in the Variable Properties section below.
- `excluded_files` - A list of files to skip over completely in the generation process.

3.2 Variable Properties

- `prompt` - The text to present to the user when asking for the value of this variable. If this value is omitted, the user won't be prompted at all, and the only way to provide a value for this variable will be on the command line.
- `default` - The default value to use if one isn't given. If this is omitted, the variable will be mandatory.
- `type` - The type of value this variable will store. Valid types are `string` (the default), `integer`, `decimal` and `boolean`.

3.3 File Names

When the skeleton is built, the name of each file and directory is processed according to a set of rules. (Note that these rules apply to files **and** directories.)

- **If a filename contains a variable name surrounded in curly brackets, it is replaced with that variable's value.**
  - Curly brackets around anything that isn't a valid variable name will cause an error.
- **If a filename contains doubled-up curly brackets, they are replaced with single curly brackets.**
  - Filenames with unmatched curly brackets will cause an error unless they're doubled-up.
- If a file's entire name is `-yorick-meta`, it is skipped over. (The reason for this, as we saw previously, is that metadata for your skeleton is stored in this directory.)

- If a filename ends with `.yorick-literal`, that bit is removed from the file name. The file name is then not processed any further, so none of the other rules on this list apply to it. (This is useful for filenames that would otherwise have special meaning to Yorick (e.g. with curly braces) or to other programs like Git (eg a `.gitignore` that is part of the template)
- Finally, if a filename ends with `.yorick-<something>` (where `<something>` is anything other than `literal`), it is processed as described in the following section, and that part of the filename is removed. Unlike `.yorick-literal`, the rest of the rules in this list do apply.

If we had a skeleton with one variable called `name`, and we constructed it setting its value to `spam`, here's how files or directories in the skeleton would be processed:

Name of file/directory in skeleton	Name of output file
`{name}.py`	`spam.py`
`{name}.py.yorick-literal`	`{name}.py`
`{{name}}.py`	`{name}.py`
`{blah}.py`	Error
`{.rst`	Error
`{.rst.yorick-literal`	`{.rst`
`{{.rst`	`{.rst`
`-yorick-meta` (directory)	directory is skipped over
`-yorick-meta.yorick-literal`	`-yorick-meta`
`eggs.yorick-literal`	`eggs`
`eggs.yorick-literal.yorick-literal`	`eggs.yorick-literal`
`name.py.yorick-j2`	`name.py` (file contents processed with `j2` engine)
`{name}.py.yorick-t`	`spam.py` (file contents processed with `t` engine)
`{name}.py.yorick-t.yorick-literal`	`{name}.py.yorick-t` (file contents copied verbatim)

3.4 File Contents

CHAPTER 4

Opening your closets to the world with Git

Closets are shared as Git repositories or tar balls, and added with the *yorick install-closet* command. They can be updated with the *yorick update-closets* command.

```
$ yorick install-closet --git http://github.com/adambrenecki/yorick-closet abre
Waiting on VCS... Done.
You can now install a skeleton from this closet by running 'yorick construct abre.<skeleton-name>'

$ yorick update-closets
Updating abre... Done.
```

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*