Linkit

Pierre Sugar
pierre@sugaryourcoffee.de

February 2, 2015

Abstract

`linkit` is a command line application that adds links to a web page that is created and updated by `linkit`. Links as said can be added but also updated and removed. It is possible to search links from the command line and list them. `linkit` can check whether the links are still active and list inactive links. Each link can have a category, tags and a discription. The category is used to group links and the discription can be searched for but tags a specifically used for searching for specific terms and topics.

1 Project outline

The programm runs from the command line. When provided a link `linkit` will check whether the link exists and adds it to the web page. A link can be anything that is accessible over an `URI`. That is web sites and files.

- add a link
- update a link
- remove a link
- check if link is alive
- search link based on description and tags
- list all links
- invoke link from command line (should open web site or file)

2 Source code management

We organize the source in `Git` at `https://github.com/sugaryourcoffee/linkit`. We first create our project with `Mix`, then we cd into the project directory, initialize our git repository, do an initial commit and push the repository to `Github`.

```
$ mix new linkit
$ cd linkit
$ git init
```

```
$ git add .
$ git commit -am "initial commit"
$ git remote add origin git@github.com:sugaryourcoffee/linkit.git
$ git push -u origin master
$
```

3 Implement the Command Line Interface

We write the test first for our command line interface to describe the functions the command line interface actually should provide. We first describe the commands of `linkit`.

```
$ linkit add "http://elixir-lang.org"\
             --tag Elixir\
             --description "Home page of Elixir"
Added "http://elixir-lang.org"
$ linkit add "http://ruby-lang.org"\
             --tag Ruby\
             --description "Home of Ruby"
$ linkit update "http://elxir-lang.org"\
             --description "Home of Elixir"
Updated "http://elixir-lang.org" with description "Home of Elixir"
$ linkit delete "http://elixir-lang.org"
Deleted "http://elixir-lang.org"
$ linkit check "http://elixir-lang.com"
Check error: "http://elixir-lang.com" is not available
$ linkit list --tag "Elixir"
URL                      | Description   | Tags
-------------------------+---------------+-------
"http://elixir-lang.org" | Home of Elxir | Elixir
$ linkit call "http://elixir-lang.org"
$ linkit list
URL                      | Description   | Tags
-------------------------+---------------+-------
"http://elixir-lang.org" | Home of Elxir | Elixir
"http://ruby-lang.org"   | Home of Ruby  | Ruby
```

The test is shown in 1 on page 2.

Listing 1: test/cli_test.exs

```
defmodule CliTest do
  use ExUnit.Case

  import Linkit.CLI, only: [ parse_args: 1 ]

  test ":help returned by option parsing with -h and --help options" do
    assert parse_args(["--help"]) == :help
    assert parse_args(["-h"])     == :help
  end
```

```
test ":help returned when no known command is given" do
  assert parse_args(["--nocommand"]) == :help
end

test ":add returned with uri, tag and description" do
  long   = ["--add", "http://example.com",
            "--tag", "tag",
            "--description", "description"]
  short  = ["--add", "http://example.com",
            "--tag", "tag",
            "--description", "description"]
  result = {:add,
            [tag: "tag", description: "description"],
            ["http://example.com"]}

  assert parse_args(long)  == result
  assert parse_args(short) == result
end

test ":update returned with uri, tag and description" do
  long   = ["--update", "http://example.com",
            "--tag", "tag",
            "--description", "description"]
  short  = ["--update", "http://example.com",
            "--tag", "tag",
            "--description", "description"]
  result = {:update,
            [tag: "tag", description: "description"],
            ["http://example.com"]}

  assert parse_args(long)  == result
  assert parse_args(short) == result
end

test ":remove returned with uri" do
  long   = ["--remove", "http://example.com"]
  short  = ["-r",       "http://example.com"]
  result = {:remove, [], ["http://example.com"]}

  assert parse_args(long)  == result
  assert parse_args(short) == result
end

test ":check returned with uri" do
  long   = ["--check", "http://example.com"]
  short  = ["-c",      "http://example.com"]
  result = {:check, [], ["http://example.com"]}

  assert parse_args(long)  == result
  assert parse_args(short) == result
```

```
  end

  test ":list returned with given tag and description filter" do
    long   = ["--list", "http://example.com",
              "--tag", "tag",
              "--description", "description"]
    short  = ["--list", "http://example.com",
              "-t", "tag",
              "-d", "description"]
    result = {:list, [tag: "tag", description: "description"],
              ["http://example.com"]}

    assert parse_args(long)  == result
    assert parse_args(short) == result
  end

end
```

The implementation so far is in 2 on page 4.

Listing 2: lib/linkit/cli.ex

```
defmodule Linkit.CLI do

  @moduledoc """
  Parses the command line and dispatches to the respective functions to add,
  update, delete, check and list URIs
  """
  @vsn 0.1

  @doc """
  Entry for parsing the command line options
  """
  def main(argv) do
    argv
    |> parse_args
  end

  @doc """
  parses *argv* that is retrieved from the command line. *argv* can have
  following values.

  * ["--help  "]
  * ["--add   ", "uri", "--tag", "tag", "--description", "description"]
  * ["--update", "uri", "--tag", "tag", "--description", "description"]
  * ["--delete", "uri"]
  * ["--check ", "uri"]
  * ["--list  ", "uri", "--tag", "tag", "--description", "description"]

  Returns tuples for each of the commands, except for help which is returned as
  an atom.
```

```
  * :help
  * {:add,    URI, TAG, DESCRIPTION}
  * {:update, URI, TAG, DESCRIPTION}
  * {:remove, URI}
  * {:check,  URI}
  * {:list,   URI, TAG, DESCRIPTION}
  """
  def parse_args(argv) do
    parse = OptionParser.parse(argv,
                               switches: [help:        :boolean,
                                          add:         :boolean,
                                          update:      :boolean,
                                          remove:      :boolean,
                                          check:       :boolean,
                                          list:        :boolean,
                                          tag:         :string,
                                          description: :string],

                               aliases:  [h: :help,
                                          a: :add,
                                          u: :update,
                                          r: :remove,
                                          c: :check,
                                          l: :list,
                                          t: :tag,
                                          d: :description])

    command(parse)
  end

  # Determine the command that has been invoked from command line and return
  # the command as an atom with the command arguments and values.
  defp command({[{:add,    true} | args], argv, _}), do: {:add,
args, argv}
  defp command({[{:update, true} | args], argv, _}), do: {:update, args, argv}
  defp command({[{:remove, true} | args], argv, _}), do: {:remove, args, argv}
  defp command({[{:check,  true} | args], argv, _}), do: {:check,
args, argv}
  defp command({[{:list,   true} | args], argv, _}), do: {:list,
args, argv}

  # When help is called command returns the atom :help. :help is also returned
  # when user invokes an unknown command
  defp command({[{:help,   true} |    _], _, _}), do:
:help
  defp command({[_                     ], _, _}), do:
:help
end
```