keepcool Documentation

Release 1

Nicolas Roche

Nov 13, 2018

CONTENTS

CHAPTER

ONE

INTRODUCTION

Here is the git repository of the project:

```
$ git clone http://www.narval.fr.eu.org/keepcool.git
```

For this documentation, we follow the main steps of the life cycle in order to improve:

- planning
- communication

Cycle de vie:

1. Expression des besoins. (see *Custommer needs*.)
2. Spécification fonctionnelle. (see *Functional specification*.)
3. Spécification technique. (see *Technical specifications*.)
4. Contrôle qualité. (see *Implementation*.)
5. Production. (see *Validation*.)
6. Obsolescence. (see *Migration*.)

CHAPTER

TWO

CUSTOMMER NEEDS

(Expression des besoins ou cahier des charges)

Dans une idée de prévenir les situations de burnout des développeurs, on voudrait produire un rapport reprenant par personne un taux de travail “hors horaires”, qu’on définirait comme le part de commits qui sont réalisés le week-end ou avant 8h / après 20h.

2.1 Notes

As not yet explicitly specified, we decided to run now under some limitations:

1. We do not take care about holidays.
2. We de not provide option to change evening and morning treshold (8 o’clock)
3. We do not score hours (2 am may be considered worse than 8:15 pm)
4. We do not take care of timezone (committers are all located in France)
5. We will build a first prototype that do not perform checks on a selected branch.
6. We are tolerant about french accents and other UTF-8 related rendering

CHAPTER

THREE

FUNCTIONAL SPECIFICATION

(Spécifications fonctionelle)

3.1 API

3.1.1 Get the list of commit

We are looking for a `python` script that take the local path of a `git`'s clone repository as parameter, and that display some computed status on committers (including the out-working-hour commit's ratio):

```
$ keepcool [OPTIONS] [GIT_PATH]
---------------------------------------
 Paserelle.git's commiter:  sum / ratio
---------------------------------------
              Josue Kouka:  112 /    3%
       Benjamin Dauvergne:  117 /   11%
            Serghei Mihai:  181 /    9%
         Frédéric Péters:  425 /   21%
             Thomas NOEL:  497 /   18%
---------------------------------------
                      all: 1395 /   15%
---------------------------------------
```

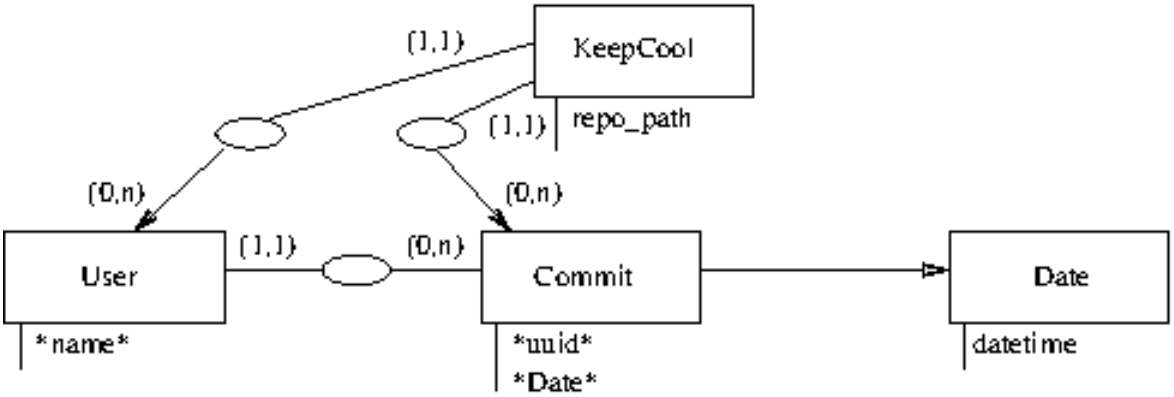
3.1.2 Options

The result should be modified by providing options:

- -h, –help: print a help message
- –version: print the project version number
- -v, –verbose: print commit's dates too
- -u, –user: limit search to a single user. ex: `"Thomas NOEL"`
- -a, –after: limit search from a date. ex: `"2018-11-09 16:05:00"`
- -b, –before: limit search up to a date. ex: `"2018-11-09 16:20:00"`
- -s, –sort: sort results on name, sum or ratio

3.2 Relational Schema

Here is the current relation schema (using Merise 2 format).



3.2.1 Notes

- We could optimise if we do not store *commits*, by computing statistics while parsing the `git` logs.

However, we keep the *commit* object and prefer to perform 3 passes (parse, compute and display) in order be keep program simplest and easy to upgrade.

- We could omit to use a key for commits as we know they are unique.

We prefer to keep the commit's uuid hash in order to provide all data, and to implement all relation in the same way (using dictionaries).

3.3 Classes

The following classes are deduced from the above relational schema.

3.3.1 Date

We need a function to check if a Unix timestamp integer belongs to normal working hours or not.

API:

```
def belongs_to_workin_hours(self):
```

Note: we can get dates by translating them using `bash`:

```
$ date --date="1978-05-11 09:45:00" "+%Y-%m-%d %H:%M:%S -> (%A %s)"
1978-05-11 09:45:00 -> (jeudi 263720700)

$ date --date="2018-11-10 01:41:14" "+%Y-%m-%d %H:%M:%S -> (%A %s)"
2018-11-10 01:41:14 -> (samedi 1541810474)
```

3.3.2 Commit

The `Commit` class inherits from the `Date`'s API.

3.3.3 Users

API:

```
def add_commit(self, commit)
def compute_status(self)
def print_status(self)
```

3.3.4 KeepCool

Main level API:

```
def get_user(self, name)
def add_user(self, name)
def add_commit(self, uuid, name, unix_timestamp)
def compute_status(self)
def print_status(self)
```

CHAPTER

FOUR

TECHNICAL SPECIFICATIONS

(Spécification technique)

4.1 Constraints

L'exercice est donc l'écriture d'un script Python qui prendrait les données du dépôt d'une de nos applications (prenons Passerelle : https://git.entrouvert.org/passerelle.git) et afficherait ces taux.

NB : on est bien sur un script simple, pas sur du web.

4.2 Library

We need a Library to interact with `git` repositories. Specifically, we need an API to parse the result of `git log`:

```
$ git log | head
commit 1d8e084753d93602aec7a6260eaa4816c3281cc2
Author: Nicolas Roche <nroche@narval.fr.eu.org>
Date:   Fri Nov 9 16:22:49 2018 +0100

    document how to build documentation
...
```

Here is a "stabilized" library:

```
# apt-get install python-git-doc
```

Unfortunately it doesn't perform any parsing, but just return a big string:

```
# apt-get install python-git
$ python
>>> import git
>>> g = git.Git('/home/nroche/git/keepcool')
>>> log = g.log()
>>> print log
...
>>> type(log)
<type 'unicode'>
```

It seems we need to play with `git` command line's option and pass it to the `log()` function.

```
$ git log --committer=nroche \
          --after="2018-11-09 16:05:00" --before="2018-11-09 16:20:00" \
          --pretty='"%cN" %ct'
"Nicolas Roche" 1541776456
"Nicolas Roche" 1541776017
$ python
>>> import git
>>> g = git.Git('/home/nroche/git/keepcool')
>>> print g.log('--pretty="%cN" %ct')
"Nicolas Roche" 1541776969
"Nicolas Roche" 1541776456
...
```

Sadly it seems to me that the only way to interact with a `git` repository is to clone it first (see no git remote access).

4.3 Logs

Not necessary as we do not develop a daemon. Traces may be sufficient.

4.4 Conclusion

- For now, we will use `python-git` library.
- We first need to clone the remote `git` repositories we want to inspect.
- We will get dates using the unix timestamp format (easier to parse)

CHAPTER

FIVE

IMPLEMENTATION

(Contrôle qualité)

5.1 Documentation

This keepcool documentation is embedded into the project:

```
# apt-get install python-sphinx
$ cd doc
$ make html
```

It is also available as PDF format:

```
$ make latex
$ cd build/latex
$ make
$ xpdf keepcool.pdf
```

Note: images should be re-compilated manually, please see *doc/README*.

5.2 Coding

We decided to follow the PEP8 coding standard:

```
# apt-get install pep8 pylint
$ pep8 keepcool.py
$ pylint keepcool.py
```

5.3 Environment

We should had use `python3` for this new project but we prefer to use the default python version of Debian Stable.

Even if still not needed, we plan to use `virtualenv` in order to work on many projects that may have different requirements on module's versions:

```
# apt-get install python-pip virtualenv
$ virtualenv dev
$ . dev/bin/activate
(dev) $
```

Troubles comes when python's modules when versions available from Debian stable are too old and cannot be find anymore usi
Because we don't need to get more up-to-date modules, we decided not to use `virtualenv` for this project.

5.4 Project structure

As I'm a newbie in writing Python project from scratch, I'm looking for an "hello world" project, like ones we can get for debian packaging:

```
$ apt-get source hello
```

I find this one but it is oriented for `django`:

```
$ git clone https://git.entrouvert.org/bidon.git/
```

I find this other one which is `pip`'s package oriented.

```
$ pip download hello-world-py
$ find .
./hello
./hello/__main__.py
./hello/__init__.py
./hello/Hello.py
./setup.py
./PKG-INFO
```

As I do not find anything convincing, I decided to use a single module file (needed for unit tests) that provide a main entry point.

5.5 Unit tests

I only use a signle file using the `unittest` framework.

```
$ ./test_keepcool.py
Ran 28 tests in 0.079s
OK
```

5.6 Code coverage

Here is the test coverage report:

```
# apt-get install python-coverage
$ ./coverage.sh
keepcool.py       ... 93%
```

5.7 Deploy infrastructure

Maybe a simple `git clone` query should be sufficient here:

```
$ mkdir repo
$ cd repo
$ git clone https://git.entrouvert.org/passerelle.git
```

However, we prefer to use `ansible` to clone and update the git repositories, as it may be useful for further operations.

```
# apt-get install ansible
$ ansible-doc git
$ ansible.sh
```

This `ansible` playbook connect localy the localhost using `ssh`, and it needs the current user il allow to connect without password:

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Note: we could also use a more recent version of ansible (but not needed here).

```
$ pip install ansible>2.4.0
```

CHAPTER

SIX

VALIDATION

(Production)

6.1 Timezone

Verify that‘‘git‘‘’s Unix timestamp match the current timezone.

```
$ git commit -a
$ date
dimanche 11 novembre 2018, 16:13:15 (UTC+0100)

$ git log --after "2018-11-11 16:00:00" --before "2018-11-11"
...
Date:   Sun Nov 11 16:11:13 2018 +0100

$ ./keepcool.py -r . --after "2018-11-11 16:00:00" --before "2018-11-11" -v
...
Date(1541949073): Sunday 2018-11-11 16:11:13 -> ['week-end']
```

6.2 Branches

Check on a repository having several branches.

```
$ mkdir linux
$ cd linux
$ git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/
$ git branch
* master
```

No branches on linux sources, I guess I miss something... (but it seems to me that `git log` returns by default the commits for all branches.)

6.3 Performances

Use a profiler tool on a udge (800.000 commits) repository.

```
$ ./keepcool.py linux/ -s 'sum'
...
 Mauro Carvalho Chehab: 29462 /   48%
    Greg Kroah-Hartman: 74367 /   70%
        Linus Torvalds: 75151 /   68%
       David S. Miller: 77947 /   74%
-------------------------------------
                  all: 796965 /   54%
-------------------------------------

$ python -m cProfile -o linux.profile keepcool.py linux
$ sudo pip install cprofilev
$ cprofilev -f linux.profile
cProfileV]: cProfile output available at http://127.0.0.1:4000

13494147 function calls (13492172 primitive calls) in 19.650 seconds
Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
746797    8.571    0.000    8.571    0.000 {built-in method poll}
796965    2.025    0.000    5.867    0.000 keepcool.py:252(add_commit)
796965    1.392    0.000    1.392    0.000 {built-in method fromtimestamp}
796965    1.053    0.000    2.446    0.000 keepcool.py:27(__init__)
     1    0.907    0.907   17.516   17.516 keepcool.py:190(parse_logs)
     1    0.693    0.693    9.933    9.933 /usr/lib/python2.7/subprocess.py:1122(_
 ↪communicate_with_poll)
746798    0.621    0.000    0.621    0.000 {posix.read}
796965    0.521    0.000    1.565    0.000 keepcool.py:74(belongs_to_workin_hours)
796965    0.499    0.000    2.945    0.000 keepcool.py:92(__init__)
...
796965    0.290    0.000    0.290    0.000 keepcool.py:230(get_user)
```

We can see that the longest times spent into functions (add_commit...), are only seven time bigger than a unique dictionary access (get_user). By the way, we should better optimise the model than the code in order to be quicker.

6.4 Memory footprint

Check the memory usage while running.

```
$ while /bin/true; do sleep 2; ps -e -ovsz -orss,args= | grep [k]eepcool; done
 25184 20196 python ./keepcool.py linux
 40384 35372 python ./keepcool.py linux
 55532 50536 python ./keepcool.py linux
 70876 65832 python ./keepcool.py linux
 87572 82612 python ./keepcool.py linux
533500 528136 python ./keepcool.py linux
653648 648432 python ./keepcool.py linux
771196 766052 python ./keepcool.py linux
376008 371328 python ./keepcool.py linux

$ while /bin/true; do sleep 2; ps -e -ovsz -orss,args= | grep [g]it; done
233292 180476 git log --encoding=UTF-8 --pretty=%H; %cN; %ct
349572 269532 git log --encoding=UTF-8 --pretty=%H; %cN; %ct
434652 356148 git log --encoding=UTF-8 --pretty=%H; %cN; %ct
500564 410912 git log --encoding=UTF-8 --pretty=%H; %cN; %ct
524036 462844 git log --encoding=UTF-8 --pretty=%H; %cN; %ct
```

The script does not use too much memory compare to the `git log` query it launches.

CHAPTER

SEVEN

MIGRATION

(Obsolescence)

This section remind all we need to know before upgrading or before to restart a new project.

7.1 List of Python modules in use

the list is given by `pylint`

```
$ pylint keepcool.py
 git
```

The internal `optparse` is deprecated, `argparse` that remplace it sould use nearly the same API.

7.2 Data

There is no output data managed or produced.

7.3 Unit tests

The unit tests may help a lot here.