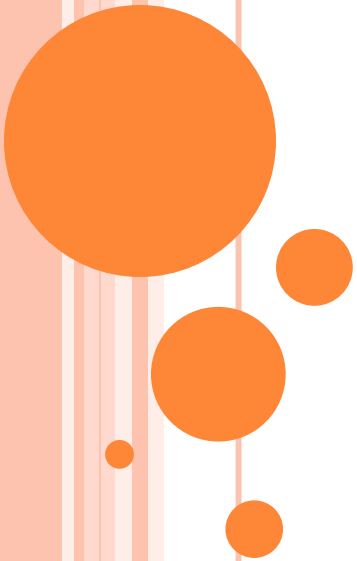


Testarium

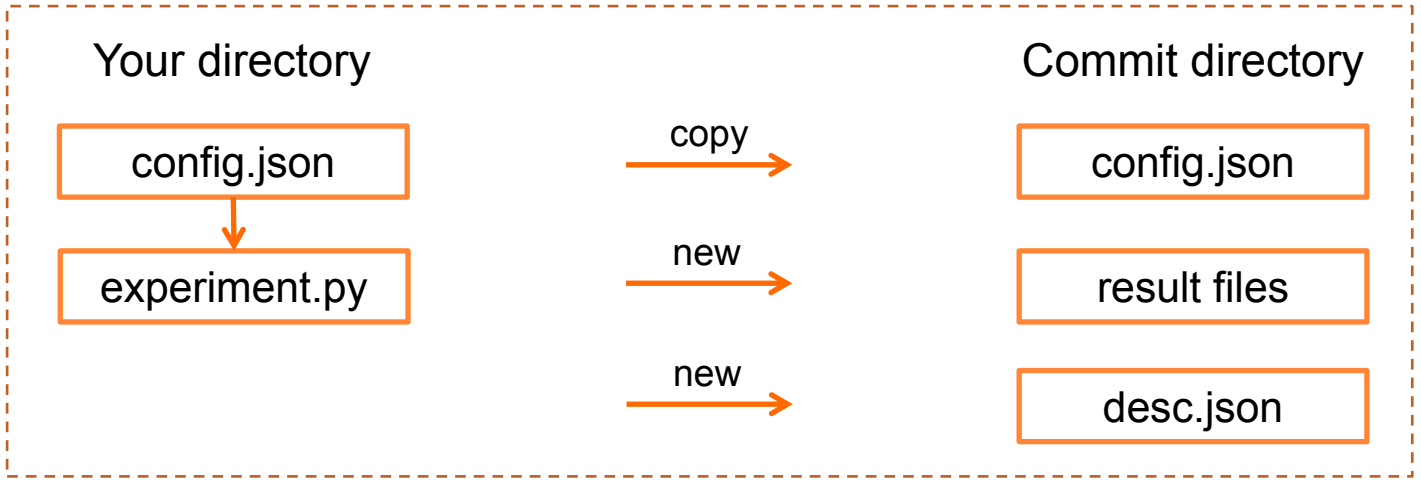
Research tool to perform experiments and store results like in the repository

Commit. What is it?

Typical experiment setup

Main testarium unit is a commit.

Each experiment run = new testarium commit



Commit. Config.json

Example:

```
{
        “float”     : 1.0,
        “bool”      : 1,
        “int”       : 10,
        “path”      : “/some/path”,
        “paramName”       : 0.5,
        “object.property”   : 42,

        "testarium.commitDirectory“:
        ".testarium/default/20141110.193532"
}
```

Testarium can add some working stuff to commit configs.

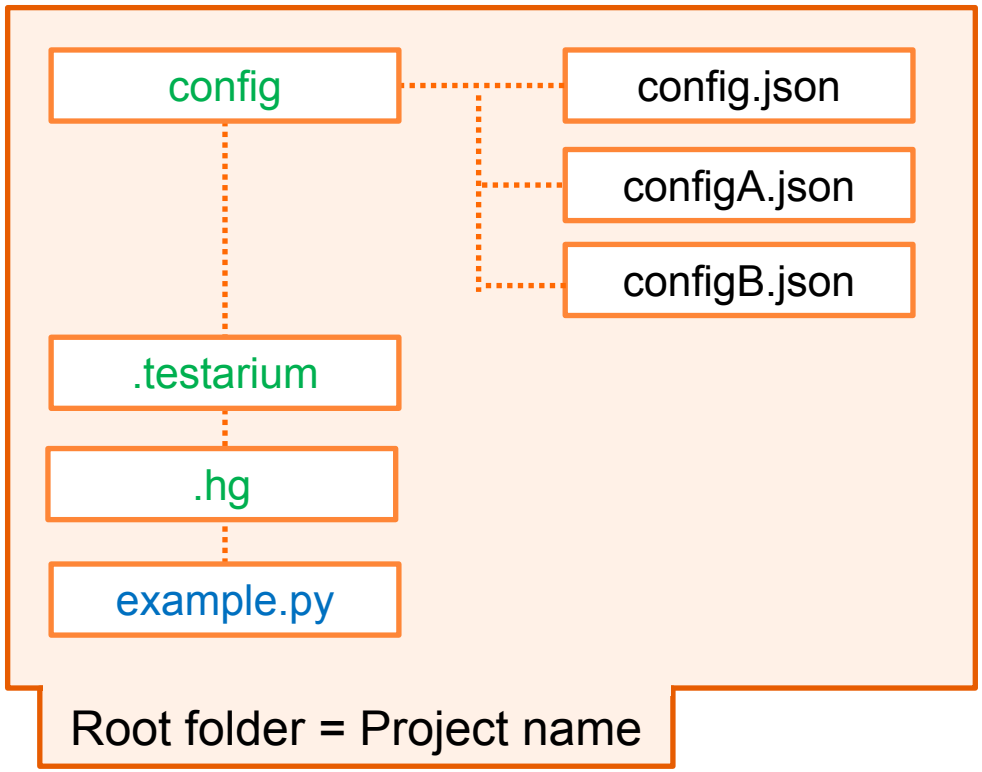
COMMIT. DESC.JSON

Example:

```
{
        "comment": “xxx",
        "name": "20141110.193532",
        "score": 0.4899,
        "params": "",
        "branch": "default",
        "duration": 0.003
}
```

Desc (or description) keeps post-experiment results, scores and other stuff. For example: minDCF or EER can be hold here.

Testarium. Project structure



default config

other config

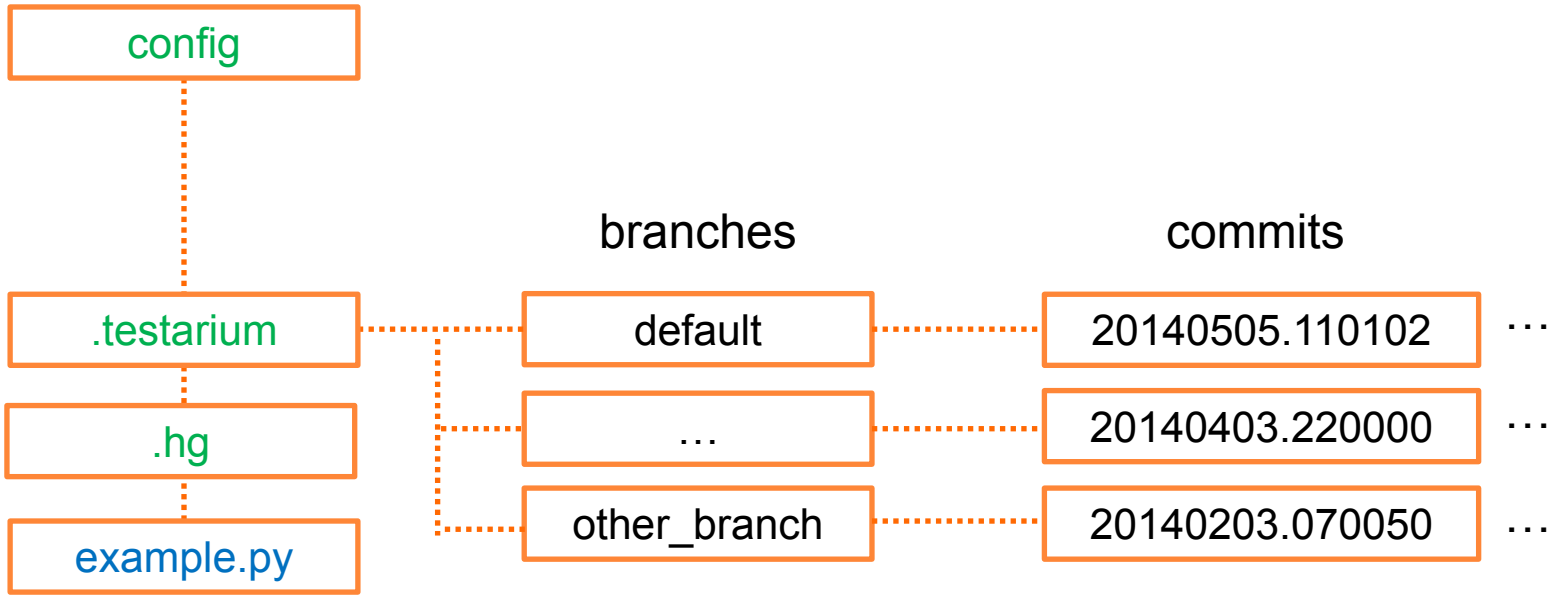
other config

testarium repository

mercurial/git repository

your project

Testarium. Inside structure



Quick start. Testarium integration

example.py

```
import testarium, testarium.score.fafr
import random, json

@testarium.experiment.set_run
def MyRun(cfg):
        c = json.loads(open(cfg, 'r').read())
        dir = c['testarium.commitDirectory']

        pos = open(dir + '/pos.txt', 'w')
        neg = open(dir + '/neg.txt', 'w')
        for i in xrange(1000): pos.write(str(random.random()) + '\n')
        for i in xrange(1000): neg.write(str(random.random()) + '\n')
        return 0

@testarium.experiment.set_score
def MyScore(commit_dir):
        return testarium.score.fafr.Score(commit_dir)
        # { 'score': 0.5 }, dict with some info, 'score' key is necessary

if __name__ == '__main__':
        testarium.testarium.best_score_is_max()
        testarium.main()
```

Quick start. Commands

- run
- branch
- del
- diff
- log
- where
- web

> example.py is equal to > example.py run

> example.py log

> example.py -h – don't forget about help

Quick start. Branches

- Default branch is ‘default’
- Active branch – current branch
- When should I use new branch?
  - different databases
  - different solutions for the same task
  - the most important results you want to store for your boss (‘release’ branch)
- Bind your source file (.py) to the specific branch:

```
if __name__ == '__main__':
        testarium.testarium.ChangeBranch(‘myBranch’)
        testarium.main()
```

> example.py branch – print current branch and config

> example.py branch new_branch new_config.json – create new

> example.py branch default – change to default branch

> example.py branch -h – don’t forget about help

Quick start. Run

> example.py (is equal to > example.py run)

run with current branch config

> example.py run new_config.json

run and use other config

> example.py run -c “my comment”

comment your commits

> example.py run -p “c[a]=1; c[b]=2”

change parameters “on the fly”

> example.py run -p “c[a]=[1,2,3]; c[b]=np.linspace(1,10)”

use search run (simple parameters combining)

Quick start. Diff

> example.py diff

show difference between best and last config

> example.py diff best 1

show difference between best config and next the last

> example.py diff 5 6

use relative the last positions, -1 is the first commit in branch

> example.py diff 20140503.120115 6

use the commit names and mix it with positions

> example.py diff best last --branchA other_branch --branchB default

best is from other_branch, last is from default branch

Quick start. Log

> example.py log (is equal > example.py log –n 5)

show descriptions of last 5 commits

> example.py log best | [0, 1, …] | 20140512.120023

show description of the best / commit at position 0,1, … or commit with name 20140512.120023

> example.py log best –n 5

show description of the best among last 5 commits

> example.py log -c -k a b

show descriptions and configs with keys ‘a’ and ‘b’ from config

> example.py log -i

show descriptions and incremental difference of configs

Quick start. Where

“Where” is something like SQL SELECT.

c – config dict, d – desc dict, you are able to use python code

“Where” is not bind with any branch, it uses all the repository commits

> example.py where “c[some]>1 and d[score]<0.2”

show commits that satisfy the condition

Like in Log:

Use -i to make incremental difference

And -c to print configs

And -k to print specified keys from config

Mercurial and git bind

If you create mercurial repository in your project directory, testarium will commit mercurial/git when “run” is called.

Testarium detects git/mercurial due to existing .git/.hg directory.

If you don’t want to make auto commits in your git/mercurial, just put this string into the .testarium/testarium.json:

```
"coderepos.use": false
```

Email Reports

General settings

```
> example.py mail --account makseq@gmail.com makseq mypassword
        --smtp smtp.server 587 --proxy 192.168.10.1 8080
        --auto [0 | 1 | … | -1]
```

Setup your mail. If you use gmail, you may not specify smtp server settings. --auto means autoreport time, time after that automatic report will be sent, use “--auto -1” to disable it.

Send forced email reports

```
> example.py run --mail
```

send email report. You should to configure general settings first.

Web Server

> example.py web -p 8080

start testarium web server on 0.0.0.0:8080

> http://localhost:8080/log?number=10

Print last 10 commits

> http://localhost:8080/log?name=last

Print last commit

> http://localhost:8080/log?name=best

Print best commit

Cookies

Advanced print

```
@testarium.testarium.set_print
def MyPrint(commit):
        try: a = str(commit.config['a'])
        except: a = ''

        score = str(commit.desc['score'])
        return ['name', 'a', 'score'], [commit.name, a, score]
```

Cookies

Advanced commits compare

```
@testarium.testarium.set_compare
def MyCompare(self, other): # self and other are commit instances
        if self._init: # commit is exist and ok
                if self.desc['score'] > other.desc['score']: return -1;
                elif self.desc['score'] < other.desc['score']: return 1;
                else: return 0
        else: return -1 # it will be used for the worst result in the best search
```