Reproducible Documentation

Release 0.1.2

Fabien C. Y. Benureau

Aug 09, 2018

Contents

CHAPTER 1

Reference

1.1 Tracking Functions

These functions can be used to track specific aspect of the computation. The tracking data will always include:

- details about the OS
- details about the CPU
- details about the Python (version, implementation)
- the command-line arguments (content of sys.argv)
- the list of installed packages (we recommend to use virtual environments to avoid unnecessary packages here)
- the timestamp (of the import of the reproducible library)

`reproducible.`**`add_repo`**(*path='.'*, *allow_dirty=False*, *allow_untracked=False*, *diff=True*)

Add a version control repository to the tracking data. Only git is supported at the moment.

Multiple repository can be tracked, each one will be identified by their path. Adding a repository twice will result in duplication of tracking data if the path strings are different.

Parameters

- **`path`** – the path to the repository. If a repository is not found there,
- **`allow_dirty`** – if False, will exit with an error if the git repository is dirty or absent.
- **`allow_dirty`** – if False, will exit with an error if the git repository is dirty or absent.
- **`diff`** – if True and uncommited changes are present in the repository, the diff will be recorded in a patchable form. Patch diffs of binary file can grow to large sizes.

Raises

- **`FileNotFoundError`** – if the path does not exist.
- **`RepositoryNotFound`** – if no repository was found.

reproducible.**add_file**(*path*, *category*, *already=False*)
Compute and store the SHA256 hash of a file, as well as its modification time (mtime).

Parameters

- **path** – the path to the file.
- **category** – group label for the file, for instance 'input', 'output', 'log', etc. Beside allowing to organize the files into categories, this is also important for the *already* flag.
- **already** – if True, will not raise ValueError if the file was already added *with the same group label*. In some workflows, an input file is also an output file later, and *reproducible* can track both state of the file, as long as they are added under different labels (presumably 'input' and 'output' in this case). If False, the existing entry, if any, will be overwritten.

Raises **ValueError** – if already is False, raise ValueError if the file was already added.

reproducible.**add_data**(*key*, *data*)
Add user-provided data to the tracking data.

If you intend to export as json or yaml, the provided key and data must be serializable in those formats.

Parameters

- **key** – label for the data. It is recommended to use a string.
- **data** – user-provided data.

reproducible.**function_args**()
Return a function's arguments value from inside the function.

The function must be called from inside the function to return the arguments from. The arguments' value will be returned as a name -> value dictionary. No difference is made between positional and provided-or-not keyword arguments. Default values of non-provided keyword arguments are included in the dictionary.

Note that this function does not add any information to the tracked context data. Use *add_data()* with the return of this function to do that.

reproducible.**add_random_state**()
Record the current random state.

The random state is different from the seed used in a *random.seed()* call. However, it can be used in the same way to produce reproducible random sequences, using the *random.setstate()* method.

record_random_state should be called just after setting the seed, and before any use of the random draw functions. Note that you can also use the current time to set the seed using *random.seed()* without an argument, and still record the random state right after.

Only the random state from the *random* module is recorded here, along with a timestamp of the recording time. If you wish to record the random state of external libraries such as *numpy*, you should use the *record_data()* method, and provide either the seed used or the result of the *numpy.random.get_state()*.

1.2 Export Functions

All export function exists in two flavor. Those that export to the disk, and those that return their output.

reproducible.**export_json**(*path*, *update_timestamp=False*)
Export the tracked data as a JSON file

Will raise error if some of the data is not JSON serializable. This method will return the SHA256 hexadecimal string of the saved file.

Parameters

- **path** – Path to the file to save the JSON data to. If the file exists, it will be overwritten.
- **update_timestamp** – The global timestamp of the tracked data, is the time of the creation of the Context instance, which happens at importing time, if using the module-level functions. If True, the timestamp will become the date of the call to *export_json*.

reproducible.**export_yaml**(*path=None*, *update_timestamp=False*)
Export the tracked data as a YAML file

Will raise error if some of the data is not YAML serializable. This method will return the SHA256 hexadecimal string of the saved file.

Parameters

- **path** – Path to the file to save the YAML data to. If the file exists, it will be overwritten.
- **update_timestamp** – The global timestamp of the tracked data, is the time of the creation of the Context instance, which happens at importing time, if using the module-level functions. If True, the timestamp will become the date of the call to *export_yaml*.

Raises **ImportError** – if the *yaml* module cannot be imported.

reproducible.**export_requirements**(*path*, *message=None*, *track_category=None*)
Export the list of installed package as a requirements.txt file.

Parameters

- **path** – The filepath to save the file, e.g: *path/to/reqs.txt* Note that no extension will be automatically included
- **message** – If not None, the message will be included after the header and before the requirements.

reproducible.**json**(*update_timestamp=False*)
Return the current tracking data, formated as JSON, as a string.

Parameters **update_timestamp** – if True, update the timestamp of the tracked data. Default False.

reproducible.**yaml**(*update_timestamp=False*)
Return the current tracking data, formated as YAML, as a string.

Parameters **update_timestamp** – if True, update the timestamp of the tracked data. Default False.

reproducible.**requirements**()
Return a list of the installed packages.

Note that if a package has been installed, changed or removed since the creation of the Context instance—or the import of reproducible when using module-level functions—this call will update the *packages* field in the tracked data.

The result is a list of strings, gathered from the results of *pip freeze*.

1.3 Git Repository Functions

The *reproducible.git_info()* and *reproducible.git_dirty()* can be used to access the state of the git repository directly.

reproducible.**git_info**(*path*, *diff=True*)
Retrieve data from the git repository.

This method can be used as a stand-alone, to access the git information directly. To add the git information to the tracked data, the *add_repo* method should be used.

Parameters **diff** – if True and uncommited changes are present in the repository, the diff will be recorded in a patchable form.

Raises

- **FileNotFoundError** – if the path does not exist.
- **RepositoryNotFound** – if no repository was found.

reproducible.**git_dirty**(*path*, *allow_untracked=False*)
Return True if the repository is dirty.

A repository is dirty if it has uncommitted changes or untracked files. This method can be used as a stand-alone, to access the git information directly. To add the git information to the tracked data, the *add_repo* method should be used.

Parameters **allow_untracked** – if False, any untracked file makes the repository dirty. Else, only uncommited changes to tracked files are considered.

Raises

- **FileNotFoundError** – if the path does not exist.
- **RepositoryNotFound** – if no repository was found.

1.4 Deprecated Functions

Those function are likely to get removed in one of the next release.

reproducible.**save_json**(**args*, ***kwargs*)

reproducible.**save_yaml**(**args*, ***kwargs*)

Index