pmpm—Python manual package manager

Release 0.2.0

Kolen Cheung

Mar 12, 2024

CONTENTS

CHAPTER

ONE

PMPM

1.1 pmpm package

pmpm—Python Manual Package Manager

1.1.1 Subpackages

pmpm.packages package

class pmpm.packages.**GenericPackage**(*env: InstallEnvironment*, *update: bool | None = None*, *fast_update: bool = False*, *arch: str = 'x86-64-v3'*, *tune: str = 'generic'*, *version: str = 'master'*)

Bases: `object`

Generic package class.

Parameters

- **env** – the environment to install the package into.
- **update** – whether to update the package if it is already installed.
- **fast_update** – whether to use fast update. If True, it will be used if the package supports it, otherwise it will fall back to normal update.
- **package_name** – the name of the package.
- **arch** – the arch to compile for.
- **tune** – the tune to compile for.
- **version** – the version to install, which should be a valid git tag/branch for git-based packages.

arch: str = 'x86-64-v3'

download() → None

env: *InstallEnvironment*

fast_update: bool = False

install_env() → None

property is_installed: bool

package_name: ClassVar[str] = ''

run_all() → None

run_conda_activated(*command: str | list[str]*, ***kwargs*) → None

Run commands with conda activated.

Parameters

kwargs – passes to subprocess.run

property src_dir: Path

property sub_platform: str

property system: str

tune: str = 'generic'

update: bool | None = None

update_env() → None

update_env_fast() → None

version: str = 'master'

Submodules

pmpm.packages.conda module

class pmpm.packages.conda.**Package**(*env:* InstallEnvironment, *update: bool | None = None*, *fast_update: bool = False*, *arch: str = 'x86-64-v3'*, *tune: str = 'generic'*, *version: str = 'master'*, *install_ipykernel: bool = True*)

Bases: *GenericPackage*

Generic package class.

Parameters

- **env** – the environment to install the package into.
- **update** – whether to update the package if it is already installed.
- **fast_update** – whether to use fast update. If True, it will be used if the package supports it, otherwise it will fall back to normal update.
- **package_name** – the name of the package.
- **arch** – the arch to compile for.
- **tune** – the tune to compile for.
- **version** – the version to install, which should be a valid git tag/branch for git-based packages.

install_env() → None

install_ipykernel: bool = True

package_name: ClassVar[str] = 'conda'

property src_dir: Path

update_env() → None

pmpm.packages.libmadam module

class pmpm.packages.libmadam.**Package**(*env: InstallEnvironment*, *update: bool | None = None*, *fast_update: bool = False*, *arch: str = 'x86-64-v3'*, *tune: str = 'generic'*, *version: str = 'master'*)

Bases: *GenericPackage*

Generic package class.

Parameters

- **env** – the environment to install the package into.
- **update** – whether to update the package if it is already installed.
- **fast_update** – whether to use fast update. If True, it will be used if the package supports it, otherwise it will fall back to normal update.
- **package_name** – the name of the package.
- **arch** – the arch to compile for.
- **tune** – the tune to compile for.
- **version** – the version to install, which should be a valid git tag/branch for git-based packages.

download() → None

install_env() → None

package_name: ClassVar[str] = 'libmadam'

property src_dir: Path

update_env() → None

update_env_fast() → None

pmpm.packages.toast module

class pmpm.packages.toast.**Package**(*env: InstallEnvironment*, *update: bool | None = None*, *fast_update: bool = False*, *arch: str = 'x86-64-v3'*, *tune: str = 'generic'*, *version: str = 'master'*)

Bases: *GenericPackage*

Generic package class.

Parameters

- **env** – the environment to install the package into.
- **update** – whether to update the package if it is already installed.
- **fast_update** – whether to use fast update. If True, it will be used if the package supports it, otherwise it will fall back to normal update.

- **package_name** – the name of the package.
- **arch** – the arch to compile for.
- **tune** – the tune to compile for.
- **version** – the version to install, which should be a valid git tag/branch for git-based packages.

property build_dir: Path

download() → None

install_env() → None

package_name: ClassVar[str] = 'toast'

property src_dir: Path

update_env() → None

update_env_fast() → None

1.1.2 Submodules

pmpm.core module

pmpm core classes.

It should implements the command line of the package manager, as well as its main logic.

class pmpm.core.**CondaOnlyEnvironment**(*prefix: ~pathlib.Path*, *file: ~pathlib.Path | None = None*, *conda_channels: list[str] = <factory>*, *conda_dependencies: list[str] = <factory>*, *pip_dependencies: list[str] = <factory>*, *dependencies: list[str] = <factory>*, *python_version: str = '3.10'*, *conda_prefix_name: str = ''*, *compile_prefix_name: str = ''*, *download_prefix_name: str = 'git'*, *conda: str = 'mamba'*, *sub_platform: str = ''*, *skip_test: bool = False*, *skip_conda: bool = False*, *fast_update: bool = False*, *install_ipykernel: bool = True*, *update: bool | None = None*, *arch: str = 'x86-64-v3'*, *tune: str = 'generic'*)

Bases: *InstallEnvironment*

Using only the stack provided by conda to compile.

Parameters

- **prefix** – the prefix path of the environment.
- **file** – the YAML file of the environment definition.
- **conda_channels** – conda channels for packages to be searched in.
- **conda_dependencies** – dependencies install via conda.
- **pip_dependencies** – dependencies install via pip.
- **dependencies** – dependencies install via pmpm.
- **python_version** – Python version to be installed.
- **conda_prefix_name** – the subdirectory within *prefix* for conda.

- **compile_prefix_name** – the subdirectory within *prefix* for compiled packages from *dependencies*.
- **download_prefix_name** – the subdirectory within *prefix* for downloaded source codes from *dependencies*.
- **conda** – executable name for conda solver, can be mamba, conda.
- **sub_platform** – such as ubuntu, arch, macports, homebrew, etc.
- **skip_test** – skip test if specified.
- **skip_conda** – skip installing/updating conda.
- **fast_update** – assume minimal change to source of compiled package and perform fast update.
- **install_ipykernel** – install this environment as an ipykernel.
- **update** – if updating all packages. If neither –update nor –no-update is provided, determine automatically.
- **arch** – -march for compilation, for example, native or x86-64-v3
- **tune** – -mtune for compilation, for example, native or generic

compile_prefix_name: str = ''

conda_prefix_name: str = ''

property environ: dict[str, str]

Return a dictionary of environment variables.

property environ_with_all_paths: dict[str, str]

Return a dictionary of environment variables with all prefixes prepended to PATH.

property environ_with_compile_path: dict[str, str]

Return a dictionary of environment variables with compile prefix prepended to PATH.

property environ_with_conda_path: dict[str, str]

Return a dictionary of environment variables with conda prefix prepended to PATH.

environment_variable: ClassVar[tuple[str, ...]] = ('CONDA_EXE', 'CONDA_PREFIX', 'HOME', 'TERM')

sanitized_path: ClassVar[tuple[str, ...]] = ('/bin', '/usr/bin')

class pmpm.core.**InstallEnvironment**(*prefix: ~pathlib.Path*, *file: ~pathlib.Path | None = None*, *conda_channels: list[str] = <factory>*, *conda_dependencies: list[str] = <factory>*, *pip_dependencies: list[str] = <factory>*, *dependencies: list[str] = <factory>*, *python_version: str = '3.10'*, *conda_prefix_name: str = 'conda'*, *compile_prefix_name: str = 'compile'*, *download_prefix_name: str = 'git'*, *conda: str = 'mamba'*, *sub_platform: str = ''*, *skip_test: bool = False*, *skip_conda: bool = False*, *fast_update: bool = False*, *install_ipykernel: bool = True*, *update: bool | None = None*, *arch: str = 'x86-64-v3'*, *tune: str = 'generic'*)

Bases: object

A Generic install environment.

Parameters

- **prefix** – the prefix path of the environment.
- **file** – the YAML file of the environment definition.
- **conda_channels** – conda channels for packages to be searched in.
- **conda_dependencies** – dependencies install via conda.
- **pip_dependencies** – dependencies install via pip.
- **dependencies** – dependencies install via pmpm.
- **python_version** – Python version to be installed.
- **conda_prefix_name** – the subdirectory within *prefix* for conda.
- **compile_prefix_name** – the subdirectory within *prefix* for compiled packages from *dependencies*.
- **download_prefix_name** – the subdirectory within *prefix* for downloaded source codes from *dependencies*.
- **conda** – executable name for conda solver, can be mamba, conda.
- **sub_platform** – such as ubuntu, arch, macports, homebrew, etc.
- **skip_test** – skip test if specified.
- **skip_conda** – skip installing/updating conda.
- **fast_update** – assume minimal change to source of compiled package and perform fast update.
- **install_ipykernel** – install this environment as an ipykernel.
- **update** – if updating all packages. If neither –update nor –no-update is provided, determine automatically.
- **arch** – -march for compilation, for example, native or x86-64-v3
- **tune** – -mtune for compilation, for example, native or generic

arch: str = 'x86-64-v3'

property compile_prefix: Path

Path to the prefix for the compiled stack by pmpm.

compile_prefix_name: str = 'compile'

conda: str = 'mamba'

property conda_bin: Path

Path to the conda binary.

conda_channels: list[str]

conda_dependencies: list[str]

conda_environment_filename: ClassVar[str] = 'environment.yml'

property conda_environment_path: Path

Path to the YAML file of the environment definition.

property conda_prefix: Path

Path to the prefix for conda.

conda_prefix_name: str = 'conda'

property conda_root_prefix: Path

Path to the root prefix of conda.

cpu_count: ClassVar[int] = 1

dependencies: list[str]

property dependencies_versioned: dict[str, str | None]

Return a dictionary of dependencies with version.

download_prefix_name: str = 'git'

property downoad_prefix: Path

Path to the prefix for the downloaded source codes by pmpm.

property environ: dict[str, str]

Return a dictionary of environment variables.

property environ_with_all_paths: dict[str, str]

Return a dictionary of environment variables with all prefixes prepended to PATH.

property environ_with_compile_path: dict[str, str]

Return a dictionary of environment variables with compile prefix prepended to PATH.

property environ_with_conda_path: dict[str, str]

Return a dictionary of environment variables with conda prefix prepended to PATH.

fast_update: bool = False

file: Path | None = None

classmethod from_dict(*data: PMPM_YAML_SPEC*) → *InstallEnvironment*

Construct an environment from a dictionary.

install_ipykernel: bool = True

property is_darwin: bool

Return True if the system is macOS.

property is_linux: bool

Return True if the system is Linux.

property mamba_bin: Path

Path to the mamba binary.

property name: str

Return the name of the environment.

pip_dependencies: list[str]

prefix: Path

python_version: str = '3.10'

run_all() → None

Run all steps to install/update the environment.

skip_conda: bool = False

skip_test: bool = False

sub_platform: str = ''

supported_systems: ClassVar[tuple[str, ...]] = ('Linux', 'Darwin')

system: ClassVar[str] = 'Linux'

property to_dict: PMPM_YAML_SPEC

Return a dictionary representation of the environment.

tune: str = 'generic'

update: bool | None = None

write_dict() → None

Write the environment definition to a YAML file.

pmpm.core.**cli**() → None

Command line interface for pmpm.

pmpm.env_variant_generator module

Generates different variants of YAML environments.

Examples:

- mkl vs. nomkl
- mpich vs. openmpi vs. nompi

pmpm.env_variant_generator.**cli**() → None

Command line interface for pmpm.

pmpm.env_variant_generator.**main**(*path: Path*, ***, *output: Path*, *mkl: bool = False*, *mpi: Literal['nompi', 'mpich', 'openmpi'] = 'nompi'*, *os: Literal['linux', 'macos'] = 'linux'*) → None

Generate the environment variants.

This is not supposed to be general-purposed, but designed only for the examples in this package.

Parameters

- **path** – Path to the YAML file.
- **output** – Path to the output YAML file.
- **mkl** – Whether to generate the MKL variant.
- **mpi** – MPI implementation to use.
- **os** – Operating system the environment is for.

pmpm.util module

pmpm.util.**append_env**(*dependencies: list[str]*, *package: str*) → None

Append a package to conda environment definition.

pmpm.util.**append_path**(*environ: dict[str, str]*, *path: str*) → None

Append to PATH in environment dictionary in-place.

pmpm.util.**check_dir**(*path: Path*, *msg: str*) → None

Check if a directory exists.

pmpm.util.**check_file**(*path: Path*, *msg: str*) → None

Check if a file exists.

pmpm.util.**prepend_path**(*environ: dict[str, str]*, *path: str*) → None

Prepend to PATH in environment dictionary in-place.

pmpm.util.**run**(*command: str | list[str]*, ***kwargs*) → None

Run command while logging what is running.

Parameters

- **command** – can be in string or list of string that subprocess.run accepts.
- **kwargs** – passes to subprocess.run

pmpm.util.**split_conda_dep_from_pip**(*dep: list[str] | list[str | dict[str, list[str]]]*) → tuple[list[str], list[str]]

Split conda and pip dependencies.

CHAPTER

TWO

REVISION HISTORY FOR `pmpm`

- v0.2.0: First PyPI release.
  - features:
    - support installing packages from another git branch / tag, such as `toast=toast3`.
    - `pmpm` cli & flags
      - replace `*_install_from_file` by `--file` flag
      - add `--pip-dependencies`
      - add `--install-ipykernel`
      - add `--arch`
      - add `--tune`
      - remove `nomkl`
    - less automatic stateful settings such as detecting Intel CPU.
    - read from / write to YAML formats rather than JSON.
    - make conda-activation works more reliably by running `conda run` instead.
    - add `pmpm_env_variant_generator`
    - refactor how defaults work.
    - drop Python 3.8-3.9, add Python 3.12.
    - remove Windows support.
  - maintenance:
    - improve type-hints.
    - refactor `pmpm.util`
    - remove `pmpm.templates`
    - add documentation on GitHub Pages and Read the Docs.
    - improve integration tests.
    - migrate from bump2version to bump-my-version
- v0.1.0: First release and proof of concept.

CHAPTER

THREE

INTRODUCTION

Python manual package manager is a package manager written in Python for manually installing a compiled stack. Before you proceed, you should know that for typical usage, `conda` (and its friends `mamba`/`micromamba`) should be enough for the purpose. `pmpm` built on top of `conda` which also compile some Python packages locally.

Goal:

- Custom compilation of some packages, possibly for optimization such as `-march` and `-mtune` which is beyond what `conda` offers.
- Provide fast re-compile after some small local changes, suitable for development.

Approaches:

- `conda_install`: Both the conda provided stack and the compiled stack from `pmpm` are in the same prefix, this makes activating an environment seamless. It is achieved by cleanly compile an environment using the conda provided stack only, including compilers.
- `system_install`: The conda provided stack and the compiled stack from `pmpm` has a different prefix. This makes the 2 completely separate, where the compiled stack can uses the host compilers. This is useful for example when the vendor provided MPI compilers are needed. This also has more points of failure, as we can't completely control what is in the host environment. `pmpm` only serves as automation for reproducibility. But you probably need to modify how `pmpm` behaves on different host, and you probably need to install packages from the OS package manager.

Alternative approach:

- You can create a conda recipe and use `conda-build` and tweak from there for customization. The only downside probably is the time it takes to re-compile after modifications.

CHAPTER

FOUR

INSTALLATION

```
pip install pmpm
```

CHAPTER

FIVE

DEVELOPMENT

```
git clone https://github.com/ickc/python-pmpm.git
cd python-pmpm
conda activate
mamba env create -f environment.yml
conda activate pmpm
pip install -e .
```

CHAPTER

SIX

USAGE

Use one of the example config in this repository:

```
pmpm conda_install "$HOME/pmpm-test" --file examples/....yml
```

CHAPTER

SEVEN

DESIGN

When installing from a YAML file such as those given in the `examples/` directory, `pmpm` behaves as a superset of conda/mamba with an extra `_pmpm` key in the YAML configuration. `pmpm` will compile packages available in `pmpm.packages` after the conda environment is created, as defined in the YAML file.

PYTHON MODULE INDEX

p

INDEX

U

V

W