smithIt CLI Documentation

Forge your project structure like a blacksmith.

smithIt is a versatile command-line tool designed to manage project structures efficiently.
This documentation provides a comprehensive guide to using smithIt, covering all available commands and their usage.

Installation

To use smithIt, you need to have Python installed on your system. You can install smithIt using pip:

```
pip install smithit
```

Usage

smithIt provides several commands to manage your project structure. Here is a detailed overview of each command:

create

Creates a new project based on a configuration file.

```
smith create [CONFIG_FILE] [OPTIONS]
```

Arguments:

- `CONFIG_FILE`: Path to the configuration file (default: smith.yaml).

Options:

- `--output, -o`: Output directory for the project.
- `--verbose, -v`: Enable verbose mode.
- `--force, -f`: Force overwrite existing project.
- `--parent, -p`: Create the parent folder.

Examples:

```
smith create
```

This will create a new project based on the default configuration file `smith.yaml` already in the current directory

```
smith create my_config.yaml
```

This will create a new project based on the specified configuration file `my_config.yaml` in the current directory.

```
smith create my_config.yaml --parent
```

This will create a new project based on the specified configuration file `my_config.yaml` in the parent directory.

```
smith create my_config.yaml --force
```

This will create a new project based on the specified configuration file `my_config.yaml` in the current directory, overwriting any existing project.

```
smith create my_config.yaml --verbose
```

This will create a new project based on the specified configuration file `my_config.yaml` in the current directory, providing verbose output.

```
smith create my_config.yaml --force --parent
```

This will create a new project based on the specified configuration file `my_config.yaml` in the parent directory, overwriting any existing project.

```
smith create my_config.yaml --force --verbose
```

This will create a new project based on the specified configuration file `my_config.yaml` in the current directory, overwriting any existing project and providing verbose output.

```
smith create my_config.yaml --parent --verbose
```

This will create a new project based on the specified configuration file `my_config.yaml` in the parent directory and providing verbose output.

```
smith create my_config.yaml --parent --verbose --force
```

This will create a new project based on the specified configuration file `my_config.yaml` in the parent directory, overwriting any existing project and providing verbose output.

```
smith create my_config.yaml --output my_project
```

This will create a new project based on the specified configuration file `my_config.yaml` in the `my_project` directory.

```
smith create my_config.yaml --output my_project --parent
```

This will create a new project based on the specified configuration file `my_config.yaml` in the `my_project` directory in the parent directory.

```
smith create my_config.yaml --output my_project --force
```

This will create a new project based on the specified configuration file `my_config.yaml` in the `my_project` directory, overwriting any existing project.

```
smith create my_config.yaml --output my_project --verbose
```

This will create a new project based on the specified configuration file `my_config.yaml` in the `my_project` directory, providing verbose output.

```
smith create my_config.yaml --output my_project --force --parent
```

This will create a new project based on the specified configuration file `my_config.yaml` in the `my_project` directory in the parent directory, overwriting any existing project.

```
smith create my_config.yaml --output my_project --force --verbose
```

This will create a new project based on the specified configuration file `my_config.yaml` in the `my_project` directory, overwriting any existing project and providing verbose output.

```
smith create my_config.yaml --output my_project --verbose --parent
```

This will create a new project based on the specified configuration file `my_config.yaml` in the `my_project` directory in the parent directory and providing verbose output.

```
smith create my_config.yaml --output my_project --force --parent --
verbose
```

This will create a new project based on the specified configuration file `my_config.yaml` in the `my_project` directory in the parent directory, overwriting any existing project and providing verbose output.

delete

Deletes specified paths.

```
smith delete [PATHS]...
```

Arguments:

- `PATHS`: List of paths to delete.

Examples:

```
smith delete path/to/file
```

This will delete the file at `path/to/file`

```
smith delete path/to/directory
```

This will delete the directory at `path/to/directory`

```
smith delete path/to/file path/to/file2
```

This will delete the files at `path/to/file` and `path/to/file2`

```
smith delete path/to/directory path/to/directory2
```

This will delete the directories at `path/to/directory` and `path/to/directory2`

```

```

```
smith delete path/to/file path/to/directory
```

This will delete the file at `path/to/file` and the directory at `path/to/directory

rename

Renames a file or directory.

```
smith rename [SRC] [DEST]
```

Arguments:

- `SRC`: Source path.
- `DEST`: Destination path.

Example:

```
smith rename old_name.txt new_name.txt
```

This will rename the file `old_name.txt` to `new_name.txt`

move

Moves files or directories to a specified destination.

```
smith move [PATHS]... [DEST]
```

Arguments:

- `PATHS`: List of paths to move.
- `DEST`: Destination directory.

Examples:

```
smith move file1.txt /path/to/destination
```

This will move the file `file1.txt` to the directory `/path/to/destination`

```
smith move folder /path/to/destination
```

This will move the directory `folder` to the directory `/path/to/destination`

```
smith move file1.txt file2.txt /path/to/destination
```

This will move the files `file1.txt` and `file2.txt` to the directory `/path/to/destination`

```
smith move folder1 folder2 /path/to/destination
```

This will move the directories `folder1` and `folder2` to the directory `/path/to/destination`

```
smith move file1.txt folder /path/to/destination
```

This will move the file `file1.txt` and the directory `folder` to the directory `/path/to/destination`

add

Adds new files or directories.

```
smith add [PATHS]...
```

Arguments:

- `PATHS`: List of paths to add.

Examples:

```
smith add new_file.txt
```

This will create the file `new_file.txt`

```
smith add new_directory
```

This will create the directory `new_directory`

```
smith add new_file1.txt new_file2.txt
```

This will create the files `new_file1.txt` and `new_file2.txt`

```
smith add new_directory1 new_directory2
```

This will create the directories `new_directory1` and `new_directory2`

```
smith add new_file.txt new_directory
```

This will create the file `new_file.txt` and the directory `new_directory`

sync

Syncs the project structure with a configuration file.

```
smith sync [OPTIONS]
```

Options:

- `--config, -c`: Config file to write the structure to (default: smith.yaml).
- `--dir, -d`: Project directory to detect structure from (default: current directory).

Examples:

```
smith sync --config my_config.yaml
```

This will write the current project structure to the configuration file `my_config.yaml`

```
smith sync --config my_config.yaml --dir /path/to/project
```

This will sync the project structure at `/path/to/project` with the configuration file `my_config.yaml`

view

Views the contents of a directory or checks if a file exists.

```
smith view [PATH]
```

Arguments:

- `PATH`: Path to view.

Examples:

```
smith view /path/to/file
```

This will check if the file `/path/to/file` exists

```
smith view /path/to/directory
```

This will display the contents of the directory `/path/to/directory`

version

Displays the version of smithIt.

```
smith version
```

Configuration File

The configuration file is a YAML file that defines the project structure. Here is an example of a configuration file:

with no templates

```yaml
project_name: MyProject
structure:
  - src:
      - main.py
      - utils.py
  - tests:
      - test_main.py
      - test_utils.py
  - README.md
  - nested_folder:
    - folder:
      - empty.txt
    - file.txt
  - empty: []
```

This configuration file defines a project with the following structure:

- `src` folder containing `main.py` and `utils.py`
- `tests` folder containing `test_main.py` and `test_utils.py`
- `README.md` file
- `nested_folder` containing a `folder` with an `empty.txt` file and a `file.txt` file
- `empty` folder with no contents

with templates (boilerplate codes)

```
project_name: MyProject
structure:
  - src:
      - main.py
      - utils.py
  - tests:
      - test_main.py
      - test_utils.py
  - README.md
  - nested_folder:
    - folder:
      - empty.txt
    - file.txt
  - temp: |
      <!DOCTYPE html>
      <html>
      <head>
          <title>MyProject</title>
      </head>
      <body>
          <h1>Welcome to MyProject</h1>
      </body>
      </html>
```

This configuration file defines a project with the following structure:

- `src` folder containing `main.py` and `utils.py`
- `tests` folder containing `test_main.py` and `test_utils.py`
- `README.md` file
- `nested_folder` containing a `folder` with an `empty.txt` file and a `file.txt` file
- `temp` file containing an HTML template

Conclusion

smithIt is a powerful tool for managing project structures. With its intuitive commands and flexible configuration, it simplifies the process of creating, modifying, and syncing project structures. Whether you're a developer, project manager, or anyone involved in project management, smithIt can streamline your workflow and enhance your productivity.