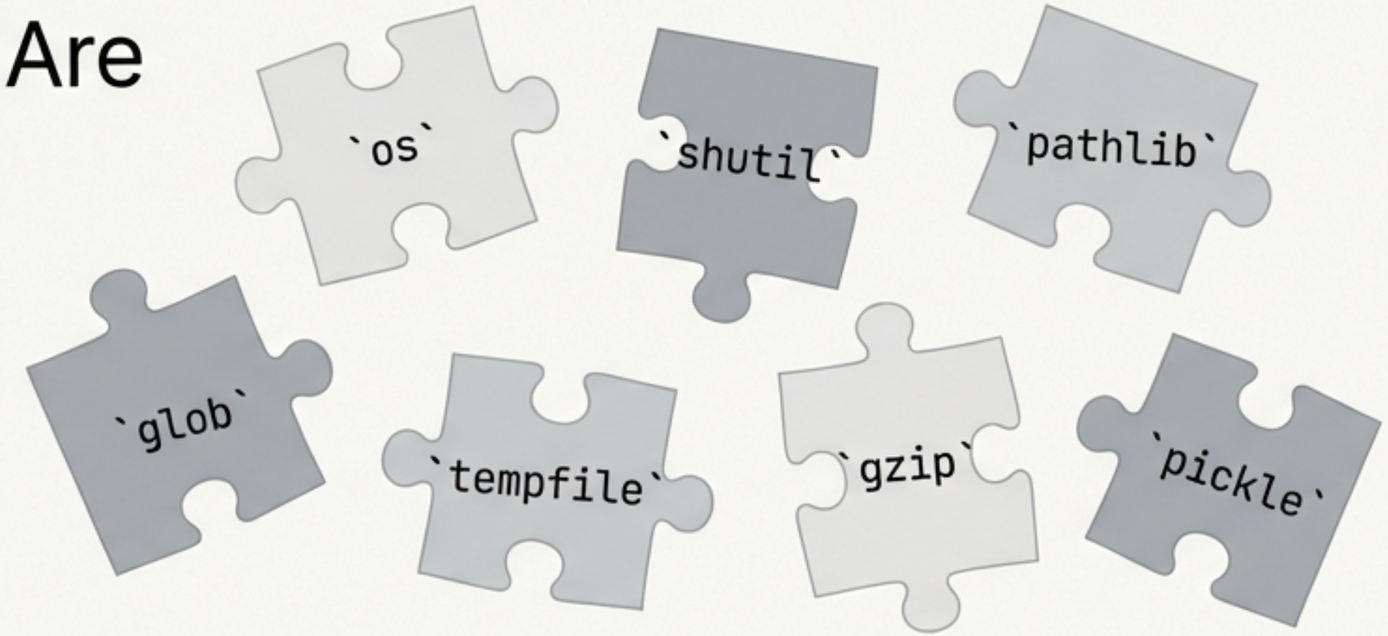
Stop Fighting File I/O in Python.

Introducing the OSBot-Utils `Files` Utility: A simple, powerful, and safe API for all your file system needs.

`pip install osbot-utils`

Python's File Operations Are Powerful, But Scattered



Inconsistent APIs: Different modules have different function names, argument orders, and return behaviours for similar tasks.

Repetitive Boilerplate: Constantly writing the same checks (`if not os.path.exists...` `os.makedirs...`) for routine operations.

Hidden Gotchas: Operations that can fail silently or require manual handling for edge cases like parent folder creation.

One Class to Rule Them All.

- **A Single, Consistent API:** Wraps the best of the standard library into one intuitive interface.
- **Eliminates Boilerplate & Friction:** Simple, readable operations that just work.
- **Your Choice of API:** Use class methods (`Files.copy`) or simple function aliases (`file_copy`).

Predictable & Safe by Design

One Job, One Function

`file_exists()` checks if a file exists. That's it. No side effects, no ambiguity.

Sensible Defaults

`temp_file()` instantly creates a unique temp file. No complex setup or manual cleanup logic required.

Automatic Safety

Operations create parent folders and check existence for you, preventing common errors.

No Surprises

Functions return useful values (like the path created or a success boolean) or `None` on failure, making your code explicit.

Write Less Code. Achieve More.

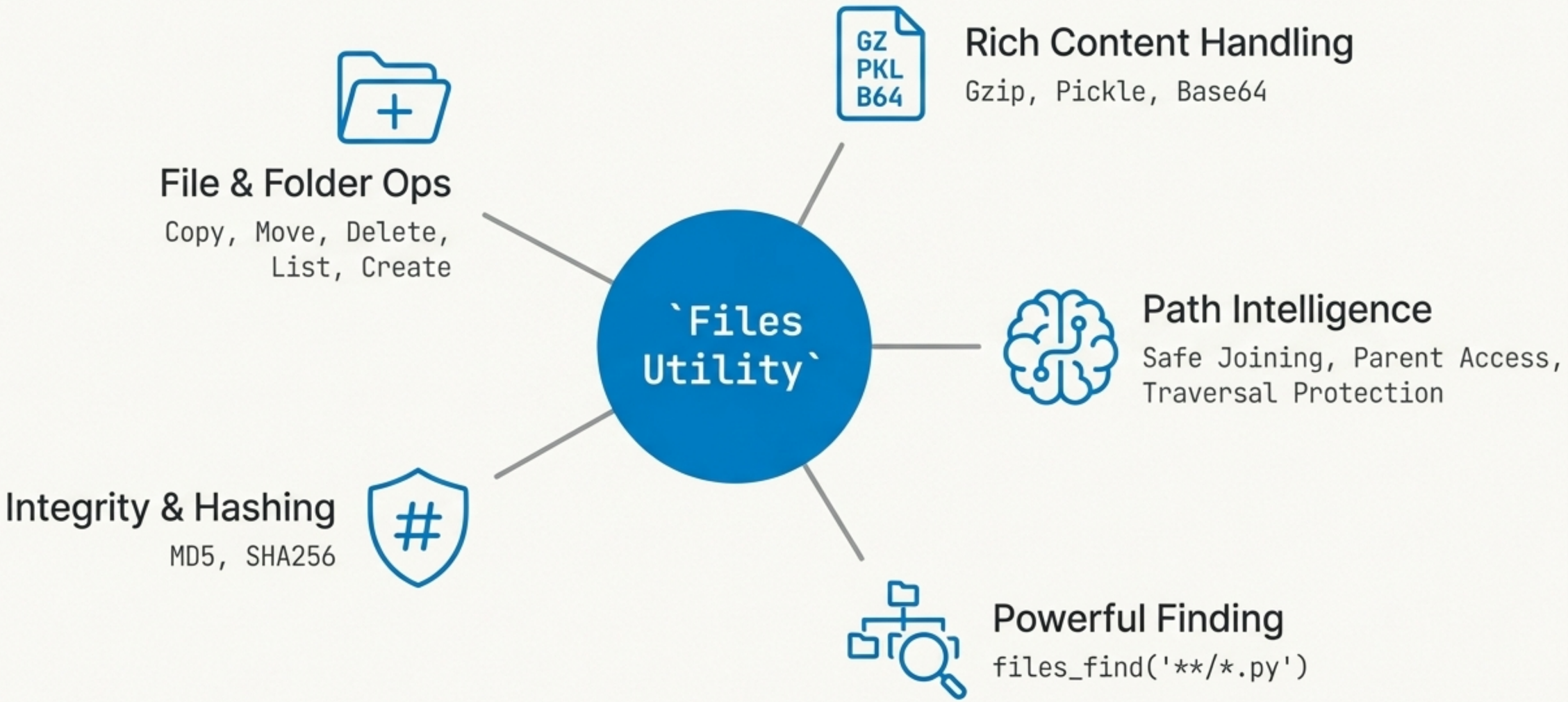
Standard Python

```python
# Safely create a file in a new subdirectory
import os

dir_path = 'path/to/new'
file_path = os.path.join(dir_path, 'file.txt')

if not os.path.exists(dir_path):
    os.makedirs(dir_path)

with open(file_path, 'w') as f:
    f.write('contents')
```

`osbot-utils`

```python
# Safely create a file in a new subdirectory
from osbot_utils.utils.Files import file_create

file_create('path/to/new/file.txt', 'contents')
```

A Complete Toolkit for Content and Operations.



Your Most Common Operations, Simplified.

Managing Files

```
file_create('path.txt', 'data')

contents = file_contents('path.txt')

if file_exists('path.txt'): ...

file_delete('path.txt')
```

Managing Folders

```
folder_create('path/to/folder')

if folder_exists('path'): ...

folder_copy('src', 'dest')

folder_delete_all('path/to/folder')
```

Temporary Items

```
tmp_file = temp_file(extension='.json')

tmp_folder = temp_folder(prefix='app_')
```

Usage Pattern: Find, Filter, and Process Files with Confidence.

A common task is to recursively find specific files in a directory tree and perform an action based on their content.

```
from osbot_utils.utils.Files import files_find, file_contains, file_name

# Find all Python files in the current project, recursively
for py_file in files_find('**/*.py', recursive=True):

    # Read and process each one efficiently
    if file_contains(py_file, 'import legacy_module'):
        print(f'Legacy import found in: {file_name(py_file)}')
```

Usage Pattern: Securely Handle User-Provided Filenames

When a user provides a filename (e.g., via an API), you must prevent **directory traversal attacks** (../.../...).

```
from osbot_utils.utils.Files import safe_file_name, path_combine_safe, file_create

user_filename = '../../etc/passwd' # Potentially malicious input
user_data = '...'

# 1. Sanitize the filename to remove path elements
safe_name = safe_file_name(user_filename) # Becomes 'passwd'

# 2. Safely combine with a trusted base directory
dest_path = path_combine_safe('/app/uploads/', safe_name)

# 3. Write the file only if the final path is valid and safe
if dest_path:
    file_create(dest_path, user_data)
    # Writes to '/app/uploads/passwd', not '/etc/passwd'
```

Quick Solutions for Common Problems.

Atomic File Write

Goal: Ensure a file write is all-or-nothing, never leaving a partially partially written file.

```
temp_path = temp_file(contents=large_data)
file_move(temp_path, final_destination)
```

Compare Two Files

Goal: Efficiently check if two files have identical contents.

```
if file_contents_md5('a.txt') == file_contents_md5('b.txt'):
    print("Files are identical.")
```

Read a Config File Safely

Goal: Read a text file, but handle the case where it doesn't exist without crashing.

```
config = file_contents('config.json')

if config is None:
    # Handle missing config file
```

Writing Robust and Readable File Code

DO

- ✓ Use `path_combine` for building paths.
- ✓ Use `path_combine_safe` for any user-influenced input.
- ✓ Explicitly handle `None` returns from functions like `file_contents`.
- ✓ Use `temp_file()` for intermediate results to avoid clutter and naming conflicts.

DON'T

- ✗ Assume a path exists before writing; `file_create` handles it, but other functions may not.
- ✗ Use simple string concatenation (`'/' + ...`) to join paths.
- ✗ Use `folder_delete_all` carelessly. It is recursive and cannot be undone.
- ✗ Ignore return values; they often indicate success/failure or provide the created path.

Common Pitfalls and Their Solutions

Problem: `file_contents` is returning `None`.

Cause: The file does not exist, or the path points to a folder. Use `file_exists()` to check first.

Problem: `folder_delete` returns `False`.

Cause: The folder is not empty. To delete a folder and all its contents, use the recursive `folder_delete_all()`.

Problem: I'm getting a `PermissionError`.

Cause: The process lacks the necessary read/write permissions for the target file or directory. This is an OS-level issue, not a library bug.

Ready to Simplify Your Code?

1. Install the Library

   ```
   pip install osbot-utils
   ```

2. Import Your Tools

   ```
   from osbot_utils.utils.Files import
   file_exists, file_create, path_combine
   ```

3. Explore the Documentation

   ```
   github.com/owasp-sbot/OSBot-Utils
   ```

Your Readability Cheat Sheet.

The library offers functional aliases for most methods. Use whichever form makes your code clearest.

Alias	Purpose
`file_create`	Create/overwrite a file with contents.
`file_contents`	Read the entire contents of a file as text.
`folder_create`	Create a folder, including any parents.
`folder_delete_all`	Delete a folder and its entire contents.
`path_combine`	Safely join path components.
`temp_file`	Create a temporary file.

Your Go-To Checklist for Frictionless File I/O

- [x] Use aliases (`file_exists` , `file_create`) for maximum readability.
- [x] Always check for existence or handle `None` returns when reading files.
- [x] Use `temp_file()` and `temp_folder()` for any scratch work or intermediate data.
- [x] Use `path_combine_safe()` and `safe_file_name()` for all user-provided path input.
- [x] Chain commands where possible: `data = file_contents(file_create(...))`
- [x] Remember hashing functions for integrity checks: `file_contents_md5()`.
- [x] Use `folder_delete_all()` with caution—it is powerful and permanent.