Automation Project

1. Project Overview

Build a **File Backup Automation Tool** using Python.

The application automatically scans a selected source folder and creates a backup of its files in another folder.

It demonstrates practical automation with:

- Folder scanning
- File copying
- Automatic backup folders
- Duplicate handling
- File statistics
- Logging
- Command line arguments

Python's `pathlib` provides cross platform filesystem path handling, while `shutil` provides high level file operations such as copying.

2. Features

- Automatic folder backup
- Recursive folder scanning
- Preserve folder structure
- Copy only new or changed files
- Create destination folders automatically
- Backup summary
- Activity log
- Command line interface
- No external packages

3. Technologies

- Python 3
- pathlib
- shutil
- argparse
- logging
- datetime

4. Folder Structure

```
file_backup_automation/
□□□ main.py
□□□ backup.py
□□□ logger.py
□□□ README.md
```

5. How the Project Works

Source Folder
    □
Scan Files
    □
Compare With Backup
    □
Copy New / Changed Files
    □
Create Backup Structure
    □
Generate Summary
    □
Save Activity Log

Example:

```
Documents/
□□□ report.pdf
□□□ notes.txt
□□□ projects/
   □□□ app.py
```

becomes:

```
Backup/
□□□ report.pdf
□□□ notes.txt
□□□ projects/
   □□□ app.py
```

6. Complete Backend Code

backup.py

```
from pathlib import Path
import shutil


class BackupManager:

    def __init__(self, source, destination):
        self.source = Path(source)
        self.destination = Path(destination)

    def validate(self):
        if not self.source.exists():
            raise FileNotFoundError(
                f"Source folder does not exist: {self.source}"
            )

        if not self.source.is_dir():
```

```
            raise NotADirectoryError(
                f"Source is not a folder: {self.source}"
            )

        if self.source.resolve() == self.destination.resolve():
            raise ValueError(
                "Source and destination cannot be the same."
            )

    def backup(self):
        self.validate()

        self.destination.mkdir(
            parents=True,
            exist_ok=True
        )

        copied = 0
        skipped = 0

        for source_file in self.source.rglob("*"):

            if not source_file.is_file():
                continue

            relative_path = source_file.relative_to(
                self.source
            )

            destination_file = (
                self.destination / relative_path
            )

            destination_file.parent.mkdir(
```

```
            parents=True,
            exist_ok=True
        )

        should_copy = (
            not destination_file.exists()
            or source_file.stat().st_mtime
            > destination_file.stat().st_mtime
            or source_file.stat().st_size
            != destination_file.stat().st_size
        )

        if should_copy:
            shutil.copy2(
                source_file,
                destination_file
            )
            copied += 1
        else:
            skipped += 1

    return {
        "copied": copied,
        "skipped": skipped
    }
```

7. Logging Code

logger.py

```
import logging


def setup_logger():
    logging.basicConfig(
```

```
        filename="backup.log",
        level=logging.INFO,
        format="%(asctime)s | %(levelname)s | %(message)s"
    )

    return logging.getLogger("backup")
```

8. Complete Application Code

main.py

```
import argparse
from datetime import datetime

from backup import BackupManager
from logger import setup_logger


def main():

    parser = argparse.ArgumentParser(
        description="Automated folder backup tool."
    )

    parser.add_argument(
        "source",
        help="Source folder to back up"
    )

    parser.add_argument(
        "destination",
        help="Destination backup folder"
    )

    args = parser.parse_args()
```

```
    logger = setup_logger()

    start_time = datetime.now()

    logger.info(
        "Backup started: %s -> %s",
        args.source,
        args.destination
    )

    try:
        manager = BackupManager(
            args.source,
            args.destination
        )

        result = manager.backup()

        logger.info(
            "Backup completed. Copied: %s, Skipped: %s",
            result["copied"],
            result["skipped"]
        )

        print("\nBackup completed successfully.")
        print(f"Files copied: {result['copied']}")
        print(f"Files skipped: {result['skipped']}")
        print(
            f"Completed at: "
            f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
        )
        print(
            f"Duration: "
```

```
            f"{datetime.now() - start_time}"
        )

    except Exception as error:

        logger.exception("Backup failed.")

        print(f"\nBackup failed: {error}")


if __name__ == "__main__":
    main()
```

9. README

README.md

```
# File Backup Automation

A Python automation tool that backs up files
from one folder to another.

## Run

python main.py "source_folder" "backup_folder"

Example:

python main.py "C:\Users\Hafsa\Documents" "D:\Backups\Documents"

## What It Does

1. Scans the source folder.
2. Finds all files.
3. Creates missing folders.
```

4. Copies new files.
5. Copies changed files.
6. Skips unchanged files.
7. Records activity in backup.log.
8. Displays a summary.

Files

main.py
Application entry point.

backup.py
Backup and file comparison logic.

logger.py
Logging configuration.

backup.log
Generated automatically after running the application.

10. How to Run

Create the project:

```
mkdir file_backup_automation
cd file_backup_automation
```

No external packages are required.

Run:

```
python main.py "source_folder" "backup_folder"
```

Example on Windows:

```
python main.py "C:\Users\Hafsa\Documents" "D:\Backups\Documents"
```

Example on macOS/Linux:

```
python main.py ~/Documents ~/Backups/Documents
```

11. Basic Testing

Create a test folder:

```
test_files/
□□□ notes.txt
□□□ report.pdf
□□□ projects/
   □□□ app.py
```

Run:

```
python main.py test_files backup
```

Expected result:

```
Backup completed successfully.
Files copied: 3
Files skipped: 0
```

Run the same command again:

```
python main.py test_files backup
```

The unchanged files should be skipped:

```
Backup completed successfully.
```

```
Files copied: 0
Files skipped: 3
```

Modify `notes.txt` and run the command again. The changed file will be copied to the backup.

12. Automation Flow

The project can later be scheduled to run automatically.

```
Scheduled Task
    ↓
Python Script
    ↓
Scan Folder
    ↓
Detect Changes
    ↓
Create Backup
    ↓
Write Log
```

On Windows, the script can be scheduled with **Task Scheduler**.

On Linux/macOS, it can be scheduled with **cron**.

13. Small Improvements

Possible next improvements:

- Automatic daily backups
- ZIP compression
- Email notifications
- Cloud storage backup

- Backup history
- Restore functionality
- File encryption
- Configuration file
- Multiple source folders
- Database for backup records
- Desktop GUI
- Scheduled backups
- Automatic cleanup of old backups

Visit haas.dev for more resources and guides.