Files and Directories

Tran Giang Son, tran-giang.son@usth.edu.vn

ICT Department, USTH

Reviews

- What is a file? What is a directory?
- What is a symlink?
- Shell commands:
  - How to list files in a directory?
  - How to show a file's content?
  - How to print all lines of a file containing a specific string?

Files

What

- Everything in UNIX is a file
- Named locations on disk to store information
  - Text file
  - Binary file

Why

Why

- RAM is volatile
  - Variables are lost after process finishes
- File is persistent
  - Data is saved

How

1. Open a file
2. Read or write
3. Close the file

How: Open a file

1. Open a file
2. Read or write
3. Close the file

- Indicates that the program wants to work with a given file
  - What file?
  - What operation to work with
- `open(fileName, mode)`
  - `fileName`: what file
  - `mode`: what operations
  - returns a `File` object representing an opened file

How: Open a file

1. Open a file
2. Read or write
3. Close the file

```
f = open("test.txt", "r")
```

Mode	Meaning
`r`	Reading (default)
`w`	Writing. Creates or clears a file.
`x`	Exclusive creation. Fails if file exists.
`a`	Appending. Creates if file does not exist.
`t`	Opens in text mode. (default)
`b`	Opens in binary mode.
`+`	Opens a file for updating (`rw`)

How: Read/write

1. Open a file
2. Read or write
3. Close the file

- `f.read(size)` reads and returns `size` bytes
  - `size` is optional
  - Reads all file content by default
  - Updates current file pointer after `.read()`
  - Be careful for large files!
- `f.seek(offset)` sets current file pointer to a specific offset
- `f.write()` writes into file

How: Read/write

```
>>> f = open("test.txt", "r+")
>>> f.read(19)
"The language's core"
>>> f.seek(0)
>>> f.read()
"The language's core philosophy is summarized in \
    the document The Zen of Python:\n* Beautiful \
    is better than ugly.\n* Explicit is better \
    than implicit.\n* Simple is better than \
    complex.\n* Complex is better than \
    complicated.\n* Readability counts.\n"
>>> f.write("That's all\n")
```

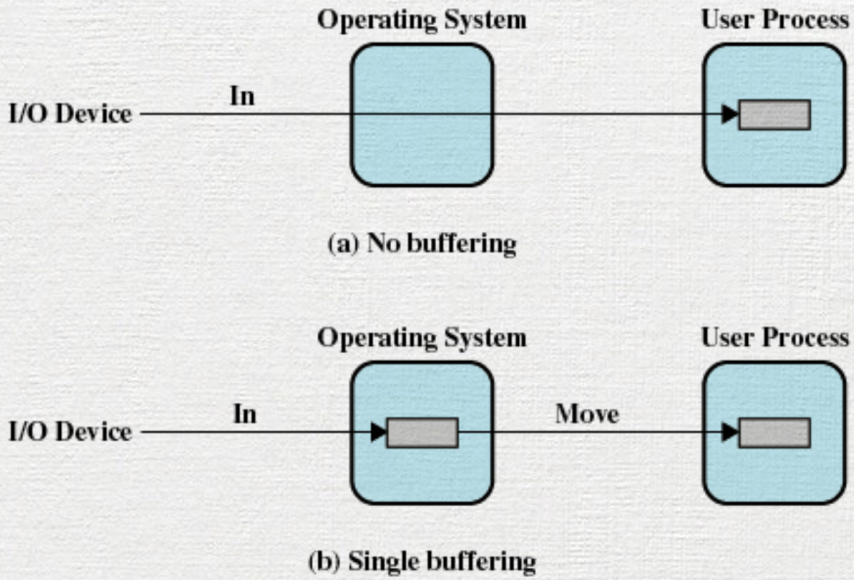
How: Read/write

- Text files
  - `.readline()`: reads until a new line.
    - There's a `\n` at the end of file
  - `.readlines()`: reads all lines

```
>>> f = open("test.txt", "r+")
>>> f.readline()
"The language's core philosophy is summarized in the documen
>>> f.readlines()
['* Beautiful is better than ugly.\n',
    '* Explicit is better than implicit.\n',
    '* Simple is better than complex.\n',
    '* Complex is better than complicated.\n',
    '* Readability counts.\n', "That's all\n"]
```

How: Buffering

- Buffer: in-memory cache of file content
  - Speeding up IO accesses[1]
  - Reading/writing blocks is faster than individual bytes



No buffering vs Single buffering

[1] Even `stdout`...

How: Buffering

- `open(fileName, mode, buffering = -1)`
- `buffering` is optional
  - `-1`, same as `io.DEFAULT_BUFFER_SIZE`
  - `0`: disable buffering
  - `1`: line buffering for text files
  - `>1`: fixed size buffer
- Flushing buffer: write buffer to disk, if any
  - Manually `f.flush()`

How: Close a file

1. Open a file
2. Read or write
3. Close the file

- Close a file after using
- Clean up OS caches, buffers
- Without closing, there may be data loss with power outage

```
f.close()
```

How: Close a file

- Automatically `.close()` using `with`

```
with open('test.txt', 'r+') as f:
    data = f.read()

# other stuffs here, f is closed.
```

How: Extras

- Exceptions
- Temporary files
- Compression
- Objects

How: Extras

- Exceptions? Remind.

1. Exceptions
2. Temporary files
3. Compression
4. Objects

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

- Exceptions? Remind.
- Exception:
  - Errors at runtime
  - Python: `try... except...`
- For handling IO errors:

```
filename = input("Enter file name: ")
try:
    f = open(filename, "r")
except IOError:
    print(f"Error missing file {filename}")
```

How: Extras

- Temporary files?

1. Exceptions
2. Temporary files
3. Compression
4. Objects

How: Extras

- Temporary files?
  - Don't care about name, location
  - Just somewhere to store temp contents
  - Automatically cleaned up after `close()`
- Module `tempfile`

```
import tempfile.TemporaryFile

# gimme a file, whenever it is
f = tempfile.TemporaryFile('w+t')
f.write("3.1415926...")
f.close()

# closed means deleted
```

1. Exceptions
2. Temporary files
3. Compression
4. Objects

How: Extras

- Compression?
  - What:

1. Exceptions
2. Temporary files
3. Compression
4. Objects

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

- Compression?
  - What: Use less storage to represent data

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

- Compression?
  - What: Use less storage to represent data
  - Why:

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

- Compression?
  - What: Use less storage to represent data
  - Why:
    - Smaller disk storage
    - Easier for transmission over network
    - Encryption with passwords

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

- Compression?
  - What: Use less storage to represent data
  - Why:
    - Smaller disk storage
    - Easier for transmission over network
    - Encryption with passwords
- Plenty of existing modules
  - `zlib`, `gzip`, `bz2`, `lzma`, `tarfile`, `zipfile`
- Each module would have different advantages/disadvantages and usage.

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

Module	Compression	In-memory	Extension	Files	Directory
`zlib`	Yes	Yes	No	No	No
`gzip`	Yes	Yes	`.gz`	No	No
`bz2`	Yes	Yes	`.bz2`	No	No
`lzma`	Yes	Yes	`.xz`	No	No
`tarfile`	No	No	`.tar`	Yes	Yes
`zipfile`	Yes	Yes	`.zip`	Yes	Yes

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

- *Serialize* objects into byte array
  - Save state to disk, optionally compressed (!)
  - Load state later
  - Transmit object to a remote machine

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

- `pickle` module
  - `pickle.dump(obj, f)`: save object `obj` into *already-opened-for-binary-write* file `f`
  - `obj = pickle.load(f)`: load object from *already-opened-for-binary-read* file `f`

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

- What can be pickled?
  - `None`, `True`, and `False`
  - Integers, floating point numbers, complex numbers
  - Strings, bytes, bytearrays
  - Tuples, lists, sets, and dictionaries containing only picklable objects
- Can also pickled behaviors:
  - Functions defined at the top level of a module
  - Built-in functions defined at the top level of a module
  - Classes that are defined at the top level of a module

How: Extras

1. Exceptions
2. Temporary files
3. Compression
4. Objects

- `pickle` vs `json`

Feature	`pickle`	`json`
Compatibility	Python-only	Open
Format	Binary	Text
Readability	Nah	Yay
Data types	Many	Limited

Directories

What

What

- Hierachical structure
  - A bunch of files
  - A bunch of sub-directories
- Looks like a tree
  - Path indicates a location inside a directory

Why

- For organization of data
- Easier traversing and browsing

How

- Listing: `os.scandir()`, `os.walk()` (recursive)
- Creating: `os.mkdir()`, `os.mkdirs()`
- Deleting: `os.rmdir()`, `shutil.rmtree()` (recursive)

How

```
>>> import os
>>> e=os.scandir(".")
>>> [f for f in e]
[<DirEntry '0. pre-intro.md'>, \
    <DirEntry '1. course intro.md'>, \
    <DirEntry '2. introz.md'>, \
    <DirEntry '3. language.md'>, \
    <DirEntry '4. oop.md'>, \
    <DirEntry '5. modules.md'>]
>>> os.mkdirs("figs/intro")
```

Practice!

Practical work 5: persistent info

- Copy your `pw4` directory into `pw5` directory
- Update your input functions
  - Write student info to `students.txt` after finishing input
  - Write course info to `courses.txt` after finishing input
  - Write marks to `marks.txt` after finishing input

Practical work 5: persistent info

- Before closing your program
  - Select a compression method
  - Compress all files aboves into `students.dat`
- Upon starting your program,
  - Check if `students.dat` exists
  - If yes, decompress and load data from it
- Push your work to corresponding forked Github repository

Practical work 6: `pickled` management system

- Copy your `pw5` directory into `pw6` directory
- Upgrade the persistence feature of your system to use `pickle` instead, still with compression
- Push your work to corresponding forked Github repository