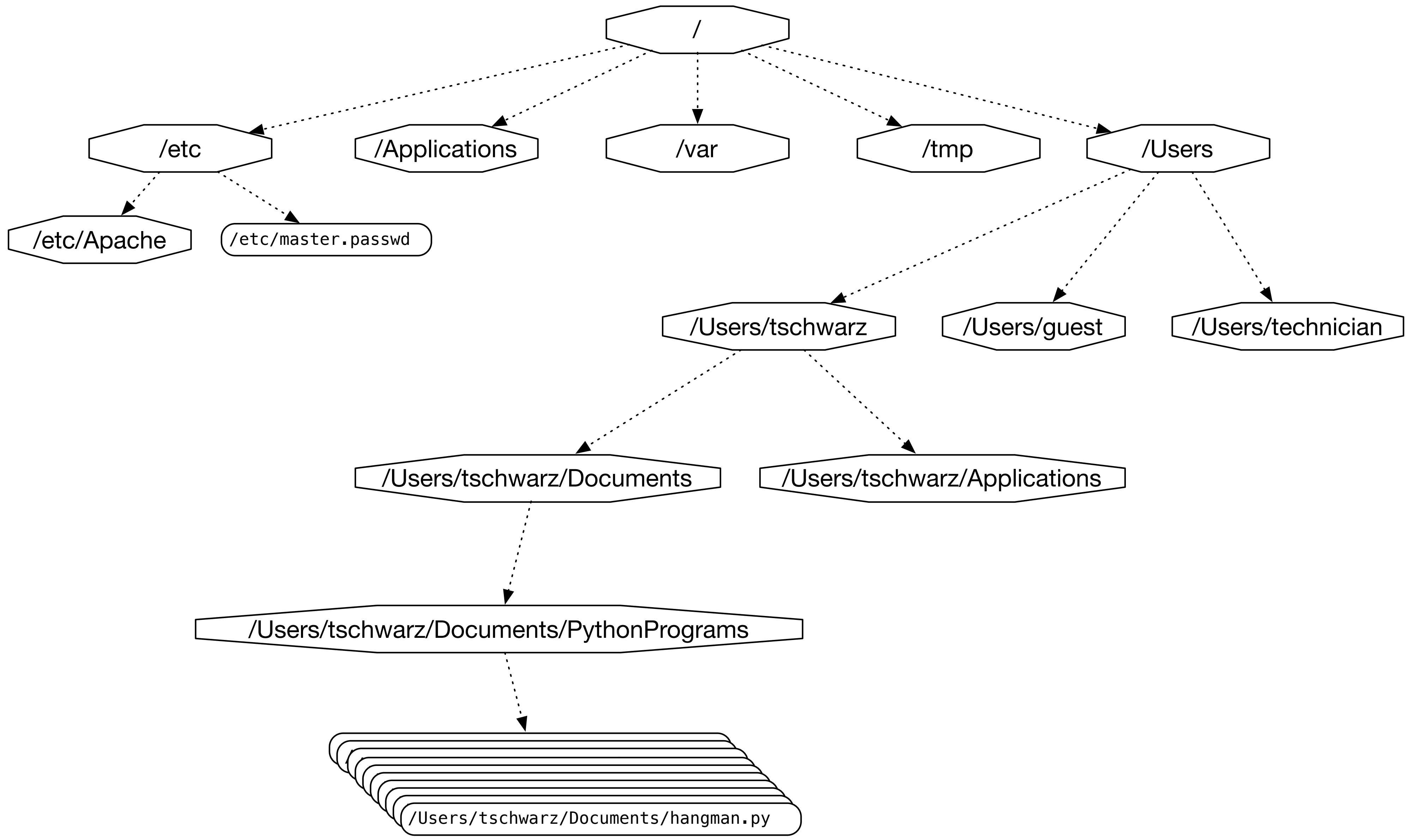
Files

Thomas Schwarz, SJ

Files

- Files
    - Basic container of data in modern computing system
    - Organized into a hierarchy of directories

Files


/
/etc
/Applications
/var
/tmp
/Users
/etc/Apache
/etc/master.passwd
/Users/tschwarz
/Users/guest
/Users/technician
/Users/tschwarz/Documents
/Users/tschwarz/Applications
/Users/tschwarz/Documents/PythonPrograms
/Users/tschwarz/Documents/hangman.py

A small subset of directories and files on a system

Files in Python

- Files accessed in
    - text mode
        - Contents interpreted according to encoding
    - binary mode
        - Contents not interpreted

Files in Python

- Python interacts by files through
  - reading
  - writing / appending
  - both

Files in Python

- Files need to be opened
  - File given by name
    - Relative path: Navigation from directory of the file
    - Absolute path: Navigation from the root of the file system

Files in Python

- File Name Examples:
    - Absolute path on a Mac / Unix

```
/Users/tjschwarzsj/Google Drive/AATeaching/Python/Programs/pr.py
```

- Relate path on a Mac / Unix
    - "../" means move up on directory

```
pr.py

../Slides/week7.key
```

Files in Python

- Windows uses backward slashes to separate directories in a file name
    - Sometimes need to be escaped: \\
    - Absolute paths need to include drive name:
        - c:\\users\\tschwarz\\My Documents\\Teaching\\temp.py
- *We will typically read and create files in the same directory as the python program is located*

Files in Python

- Before files are used, program needs to open them
- After they are being used, program should close them
    - Will automatically closed when program terminates
    - Long-running programs could hog resources

Opening Files in Python

- File objects have normal variable names

```
inFile = open(“data.txt”,”w”)
```

- opens a file “data.txt” in write mode

- open takes :
  - file name — absolute / relative path
  - mode — r (read), w (write), a (appending)
  - mode — b (binary), “” (not binary)

Closing Files in Python

- We close file by invoking close
    - `inFile.close()`

Why we need to close files

- Files are automatically closed when the program terminates
- When one application has opened a file for writing it acquires a write lock on the file and no other application can access the file.
- When one application has opened a file for reading, it acquires a read lock on the file and no other application can write to it.
- If you write programs that last more than a few seconds, you do not want to hog files when you do not need them.
- **There is no guarantee that an open file has seen all changes**

With-clauses

- Python 3 allows us to open and close files in a single block (context)

```
with open("twoft8.11.txt") as inFile, open("twoftres8.11.txt",
"w") as outFile:

        #Here you work with the file
```

Processing Files in Python

- We write strings to the file

```
with open(‘somefile.txt’,’wt’) as f:
 f.write(str(500)+”\n")
```

- Redirect print

```
with open(‘somefile.txt’,’wt’) as f:
 print(500, file = f)
```

Processing Files in Python

- Reading files
  - The read-instruction

    ```
    string = inFile.read(10)
    ```

    reads ten bytes of the file
  - Read the entire file

    ```
    with open(‘somefile.txt’, ‘rt’ as f:
        data = f.read()
    ```

Processing Files in Python

- Reading files
  - Read line by line

```
with open('somefile.txt', 'rt') as f:
    for line in f:
        #process line
```

More String Processing

- To process read lines:
  - `strip()` and its variants `lstrip(), rstrip()`
    - Remove white spaces (default) or list of characters from the beginning & end of the string
  - `split()` creates a list of words separated by white space (default)

```
"This is a sentence with many words in
it.”.split()
```

```
['This', 'is', 'a', 'sentence', 'with',
'many', 'words', 'in', 'it.']
```

Examples

- Finding all words over 13 letters long in “Alice in Wonderland”
  - Download from Project Gutenberg

```
import string

with open("alice.txt", "rt", encoding = "utf-8") as f:
    for line in f:
        for word in line.split():
            if len(word) > 13:
                print(word)
```

Examples

- Count the number of words and of lines in “Alice in Wonderland”
    - Read the file line by line
        - The number of words in a line is the length of line.split.

```
import string

line_counter = 0
word_counter = 0
with open("alice.txt", "rt", encoding = "utf-8") as f:
    for line in f:
        line_counter += 1
        word_counter += len(line.split())
print(line_counter, word_counter)
```

Problems with Line Endings

- ASCII code was developed when computers wrote to teleprinters.
  - A new line consisted of a carriage return followed or preceded by a line-feed.
- UNIX and windows choose to different encodings
  - Unix has just the newline character   “\n”
  - Windows has the carriage return: “\r\n”
- By default, Python operates in “universal newline mode”
  - All common newline combinations are understood
  - Python writes new lines just with a “\n”
- You could disable this mechanism by opening a file with the universal newline mode disabled by saying:
  - `open(“filename.txt”, newline=‘’)`

Encodings

- Whenever you see strings:
    - Think about encoding and decoding
        - Example: the ë
            - `'ë'.encode('utf-8').decode('latin-1')`
        - gives
            - `'Ã«'`
- Mixing encodings often creates chaos

Encodings

- Python is very good at guessing encodings
  - Do not guess encodings
    - E.g.: Processing html: read the http header:
      - `Content-Type: text/html; charset=utf-8`
  - If you need to guess, there is a module for it:
    - `chardet.detect(some_bytes)`

Encodings

- Thinking about encoding and decoding string allows easy internationalization