COMP1521 Week 8

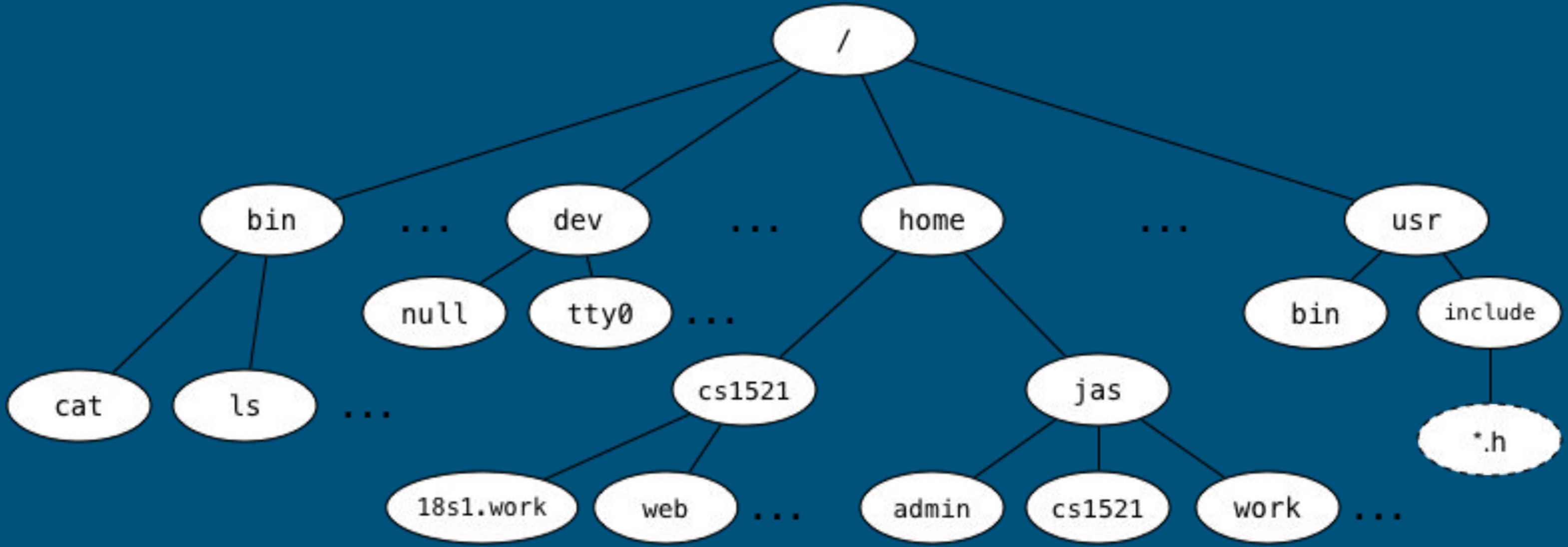
(Week 1 of) Intro to Operating Systems

Assignment 2 is out!

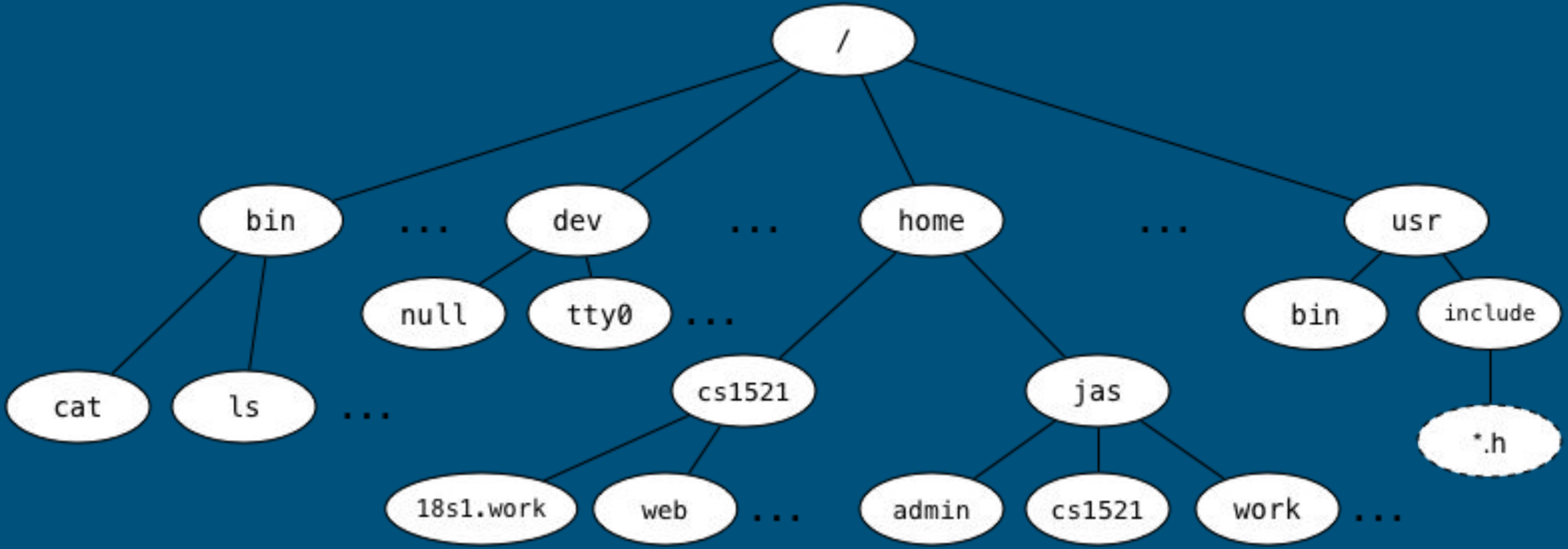
- Due Week 11 Monday at 5pm.
- Focused on file operations and bit operations.
- Assignment divided into stages - early stages relatively straightforward, later stages get more difficult.
- Would recommend starting early!

Unix file philosophy

Unix filesystem (tree-structured)

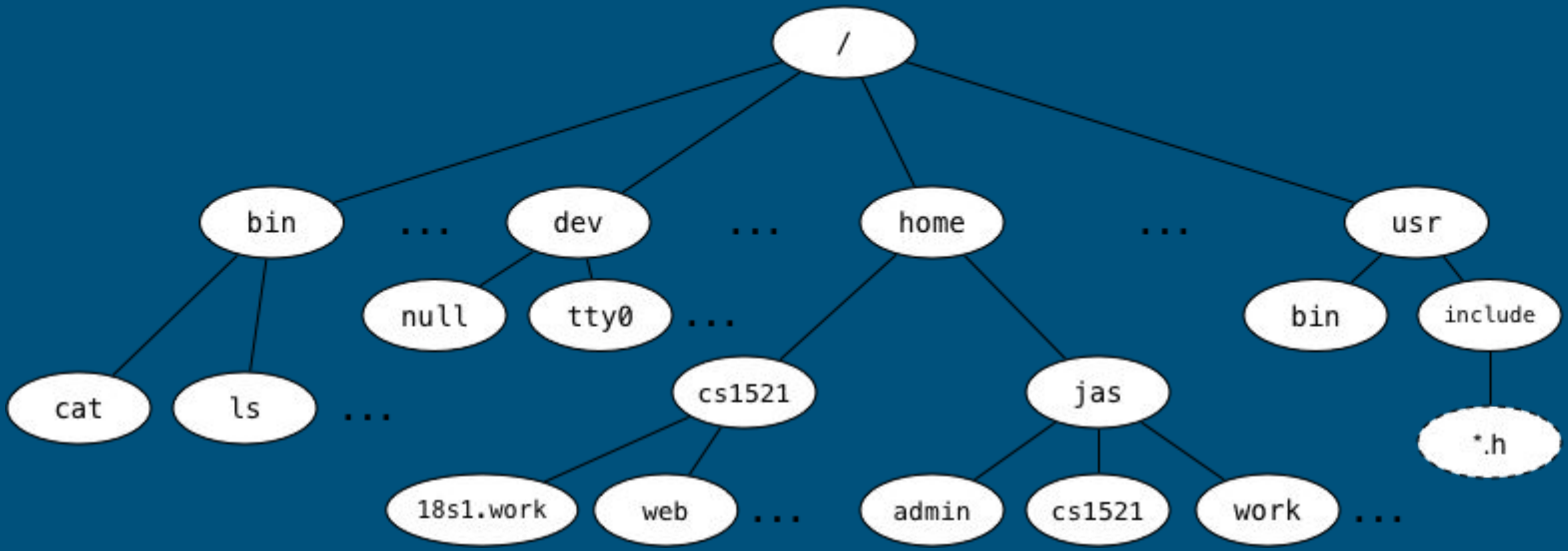


Q: What is the full pathname of COMP1521's web directory?



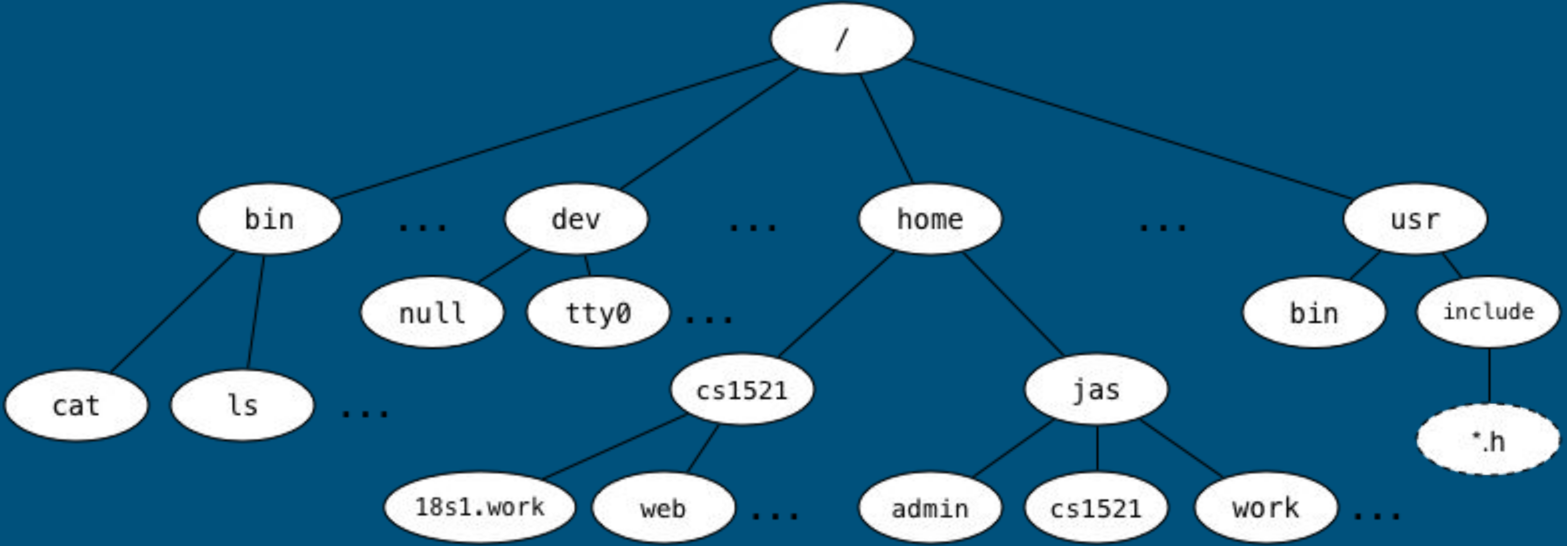
A: /home/cs1521/web

Which directory is ~jas/../..



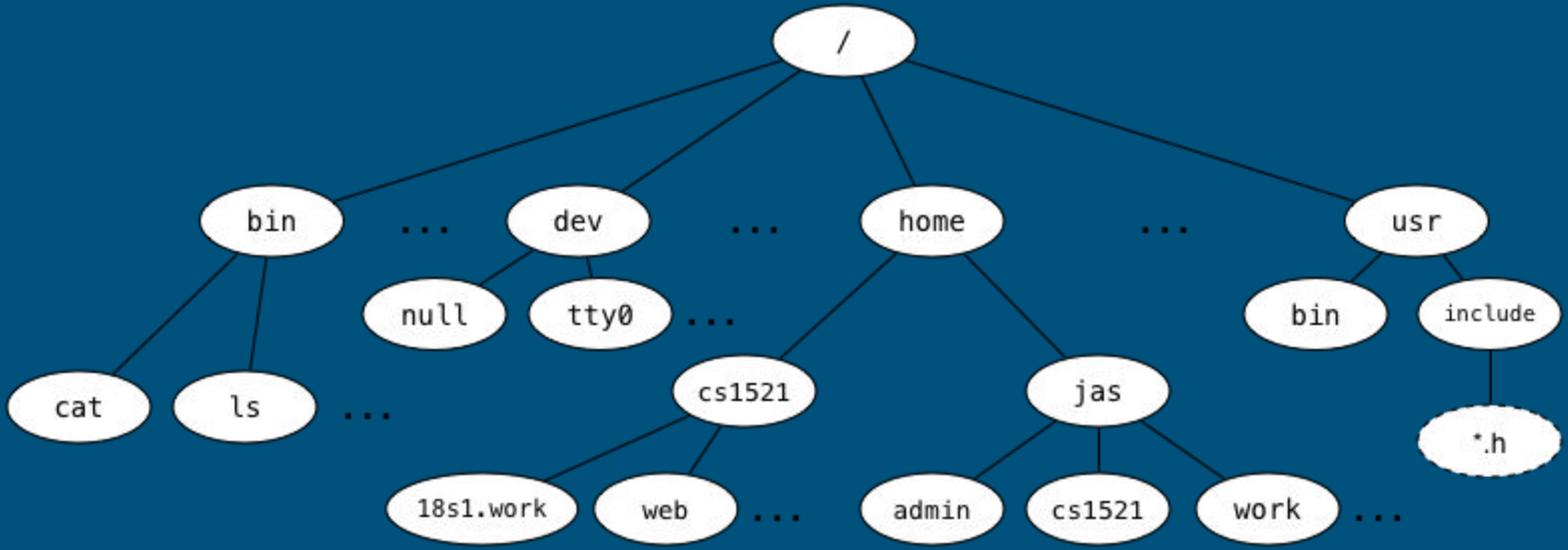
A: ~jas/../.. is the root directory (/)

Links to the children of a given directory are stored as entries in the directory structure. Where is the link to the parent directory stored?



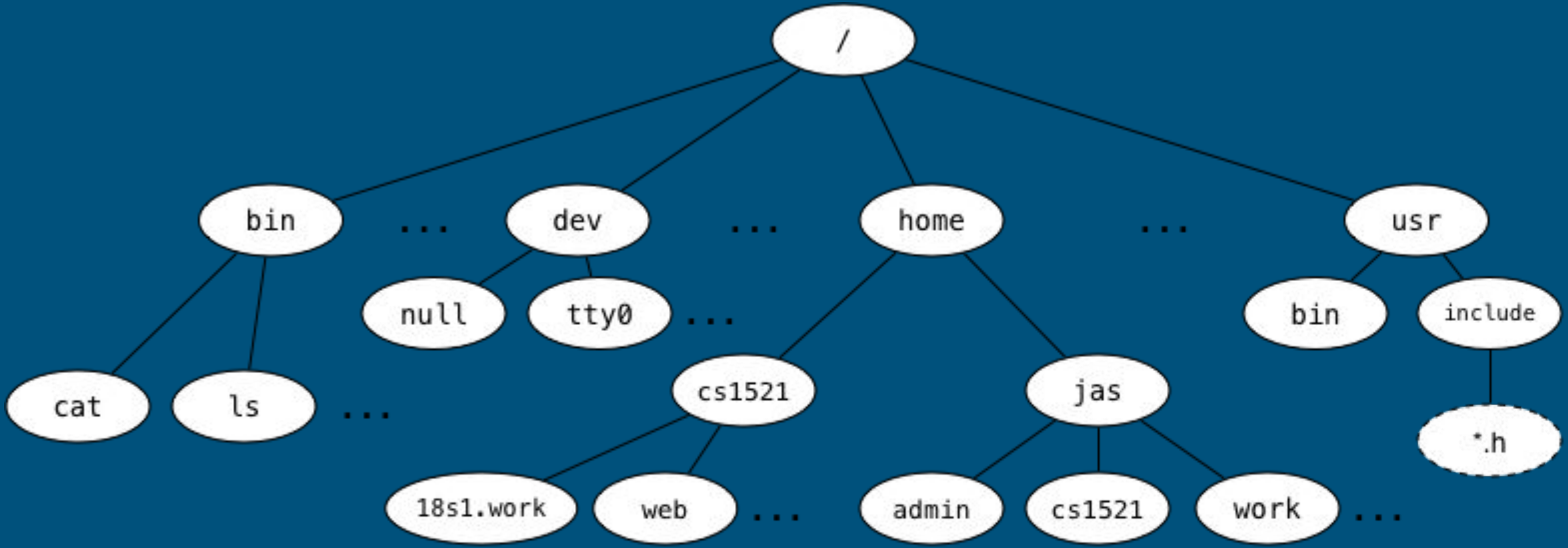
A: The link to the parent directory is also stored as an entry in the directory structure, called ..

What kind of filesystem object is cat?



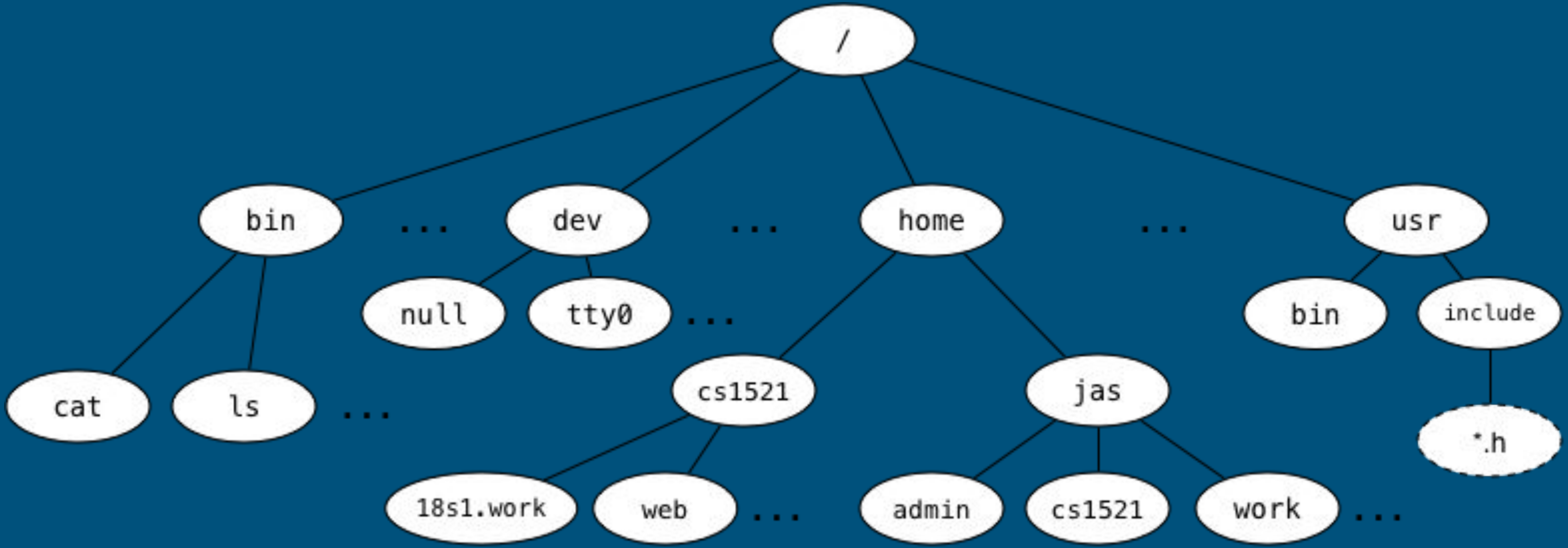
A: cat is a regular file, which happens to contain executable machine code.

What kind of filesystem object is home?



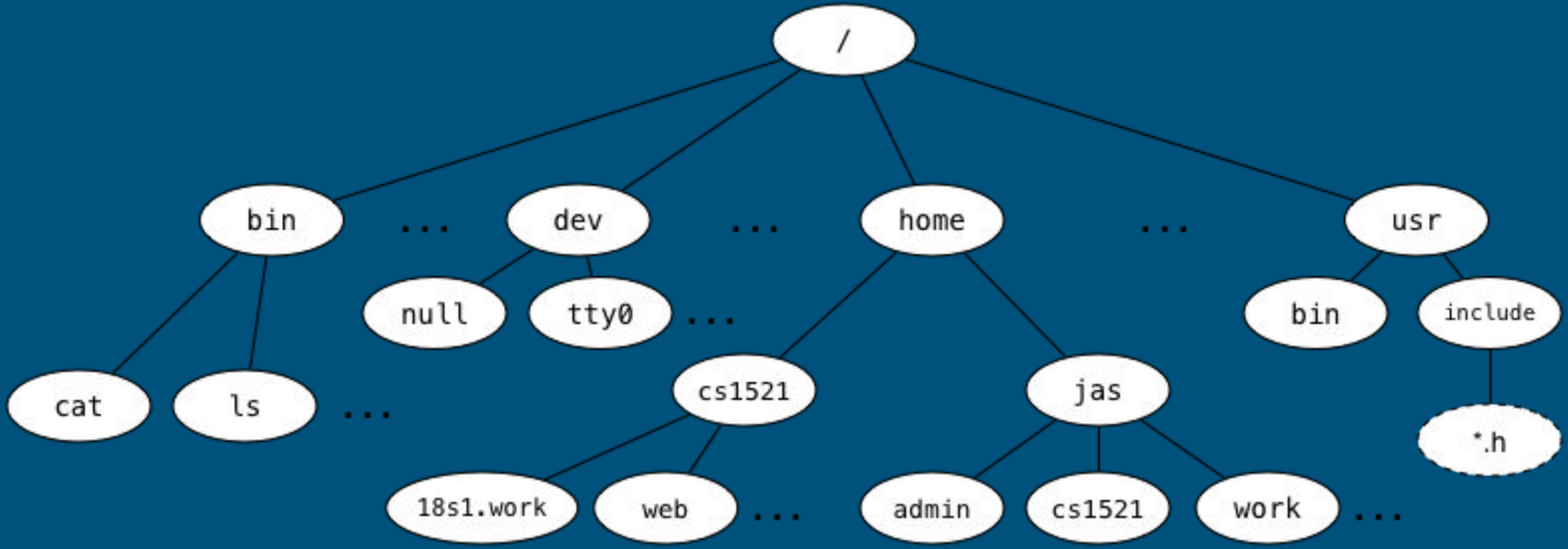
A: a directory.

What kind of filesystem object is tty0?



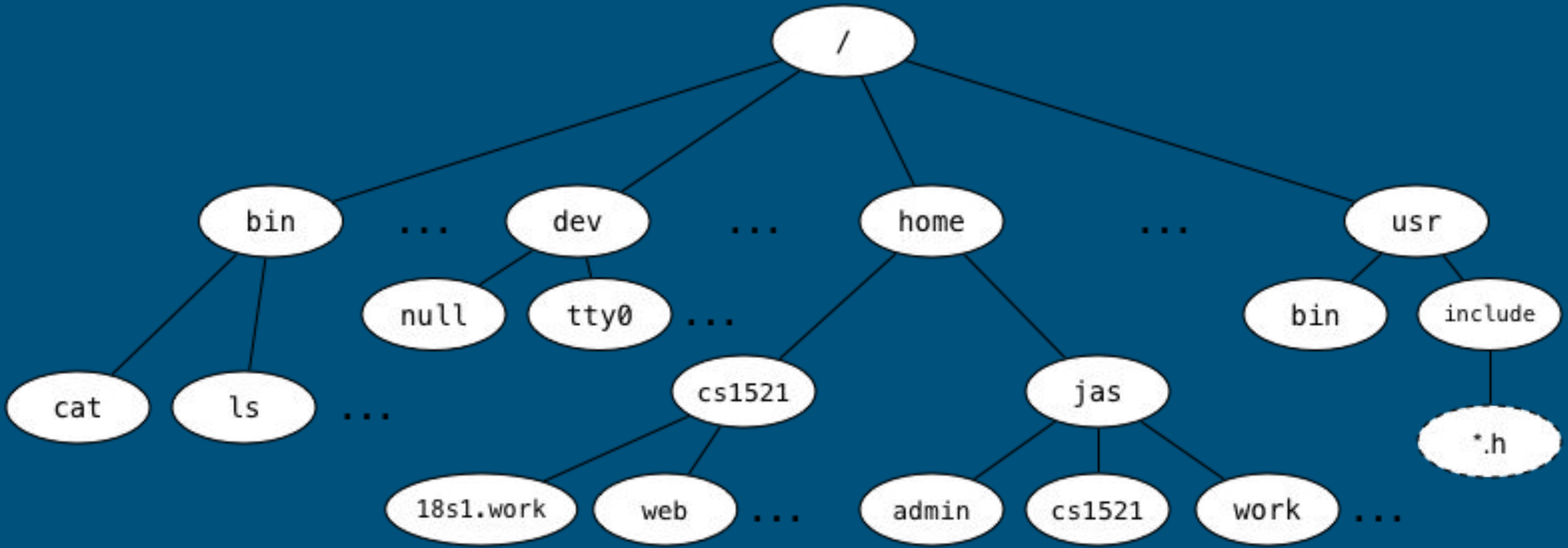
A: tty0 is a character special file, a file that provides access to an input/output device (i.e. some kind of byte stream).

What kind of filesystem object is a symbolic link? What value does it contain?



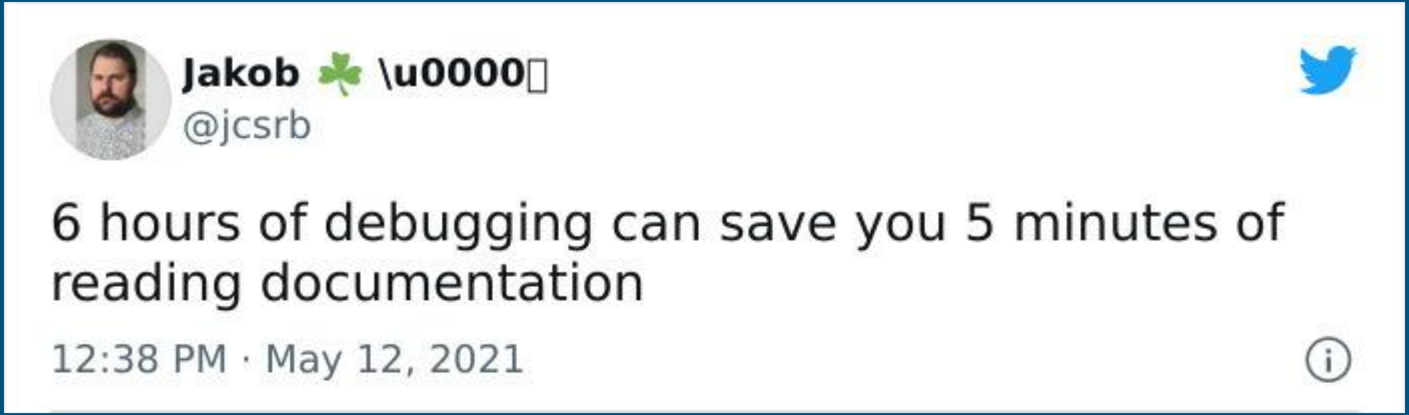
A: A symbolic link is a special kind of file that simply contains the name of another file.

Symbolic links change the filesystem from a tree structure to a graph structure. How do they do this?



A: Because of the parent-child relationship of the tree structure, we can’t really have a graph without symlinks, since an object can only have one parent.

Time for code!



Jakob ☘️ \u0000⌷
@jcsrb

6 hours of debugging can save you 5 minutes of reading documentation

12:38 PM · May 12, 2021

All about fopen

FILE *fopen(const char *pathname, const char *mode);

- pathname is the name of the file to be opened. Can be **relative** or **absolute**.
- Mode describes what we are opening the file for
  - “r” for reading.
  - “w” for writing (will overwrite the file if it already exists…also creates the file if it doesn’t exist).
  - “a” for appending (will also create the file if it doesn’t exist).
- Returns NULL if it fails

Why could fopen fail

- If the file you tried to open for reading does not exist.
- If you try to open a file you do not have permission to access.
- If the "mode" string was invalid.
- If the system is out of memory.
- If the pathname was too long (linux max is 4096 bytes).

More code!

Why shouldn’t you use fgets/fputs with binary data?

- Binary data could have zero bytes in it (i.e. 0x00). This is normally written as ‘\0’ in a C string, and should look familiar! It’s the null terminator.
- These functions will consider this as the end to the “string” when you really intend to, for example, write some data.
- We use fgetc/fputc instead.

Quick reminder on characters.

- We generally use ascii to represent characters in C.
- ASCII is a nice way of representing characters within one byte. However, it has some quirks that can lead to common bugs.
- For example, the character 1 does not equal the number one
    - i.e. ‘1’ != 1
    - Instead, ‘1’ == 49
- Type “man 7 ascii” to access the ASCII table in a terminal.
- What does the line “*printf ("%c%c%c%c%c%c", 72, 101, 0x6c, 108, 111, 0x0a);*” produce?

How many different values can fgetc return?

- 256?
- After all, a byte can store 2^8 = 256 values…
- That would include values (0-255) (read as unsigned values).
- However, that’s not all fgetc can return! It can also return EOF, which is generally defined as -1.
- So it can return 257 values!
- This is why using a char to store the result of fgetc can result in bugs…

Why are the names of fgetc, fputc, getc, putc, putchar, and getchar misleading?

- Because they really deal with bytes and not characters.
- These functions don’t have to strictly deal with data that is ASCII, for example…they can be used to interact with any binary data!
- It has been suggested that replacing the ‘c’ with a ‘b’ (e.g. fgetb) would be clearer.

lseek(int fd, off_t offset, int whence)

- What do we use it for?
- Moving the position of a file descriptor.
- Useful when the internals of a file are in a well-established format
    - For example, records of a consistent length
- Return value returns file position after the lseek is done. Returns -1 if it fails.
- fseek exists as a more portable version.
- “whence” takes 3 values
    - SEEK_SET (seek from file start)
    - SEEK_CUR (seek from current position)
    - SEEK_END (seek from file end)