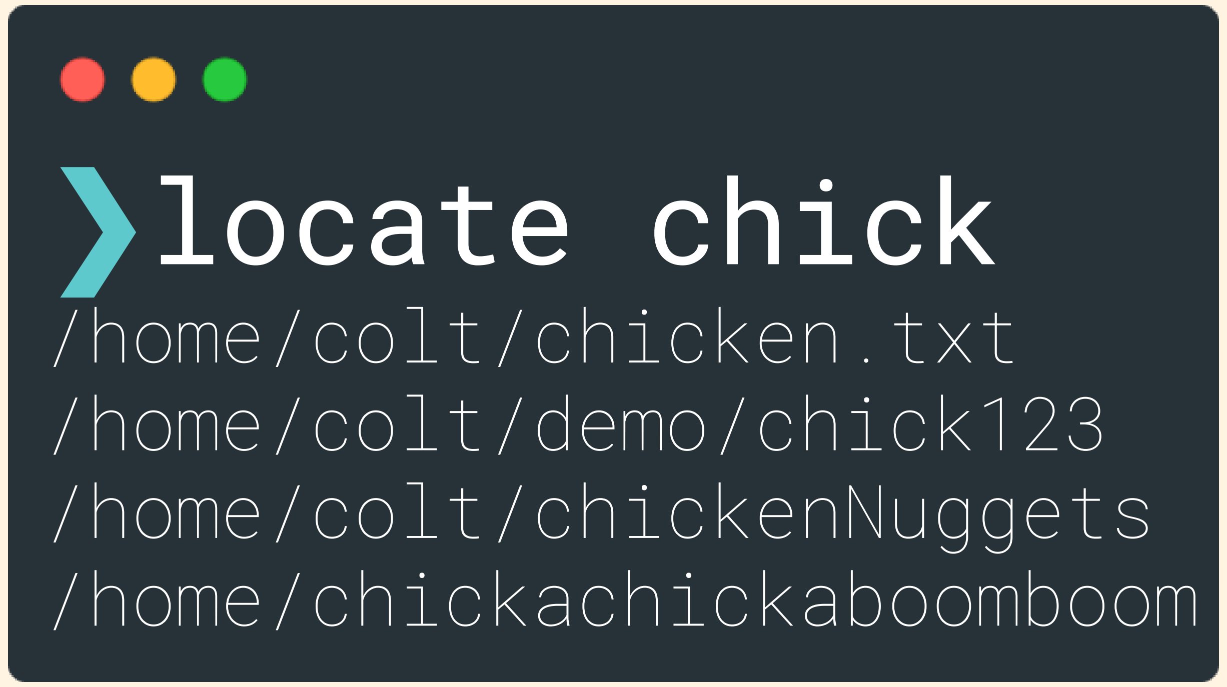
Finding Things

locate

The **locate** command performs a search of pathnames across our machine that match a given substring and then prints out any matching names.

It is nice and speedy because it uses a pre-generated database file rather than searching the entire machine.

For example, **locate chick** will perform a search for all files that contain chick in their name.



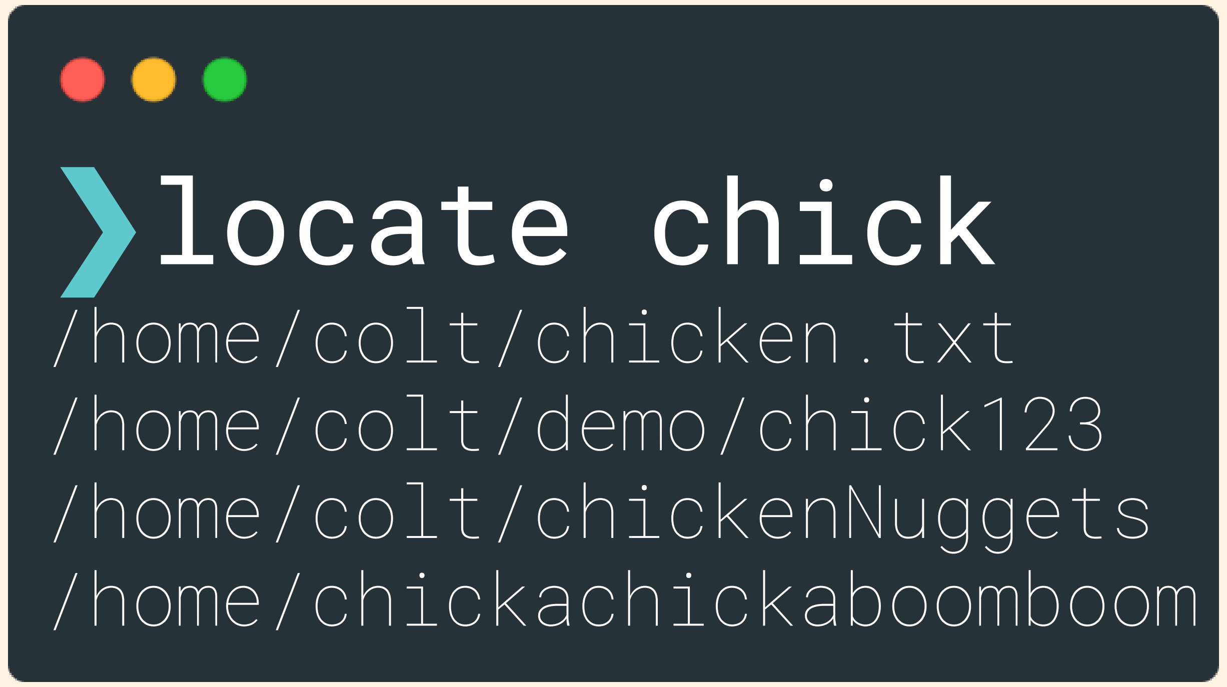
```
❯locate chick
/home/colt/chicken.txt
/home/colt/demo/chick123
/home/colt/chickenNuggets
/home/chickachickaboomboom
```

locate options

The **-e** option will only print entries that actually exist at the time locate is run.

The **-i** option tells locate to ignore casing

The **-l** or **--limit** option will limit the number of entries that locate retrieves.



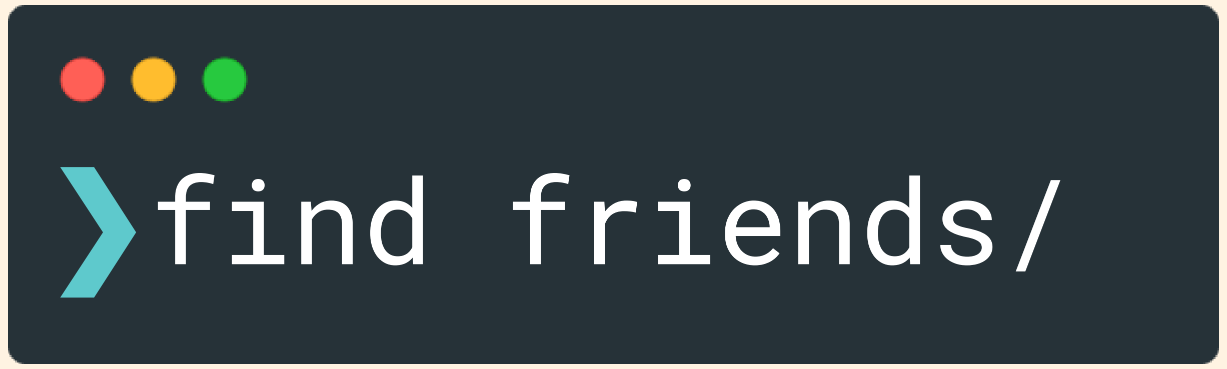
```
❯locate chick
/home/colt/chicken.txt
/home/colt/demo/chick123
/home/colt/chickenNuggets
/home/chickachickaboomboom
```

find

The locate command is nice and easy, but it can only do so much! The **find** command is far more powerful! Unlike locate, find does not use a database file.

By default, **find** on its own will list every single file and directory nested in our current working directory.

We can also provide a specific folder. **find friends/** would print all the files and directories inside the friends directory (including nested folders)



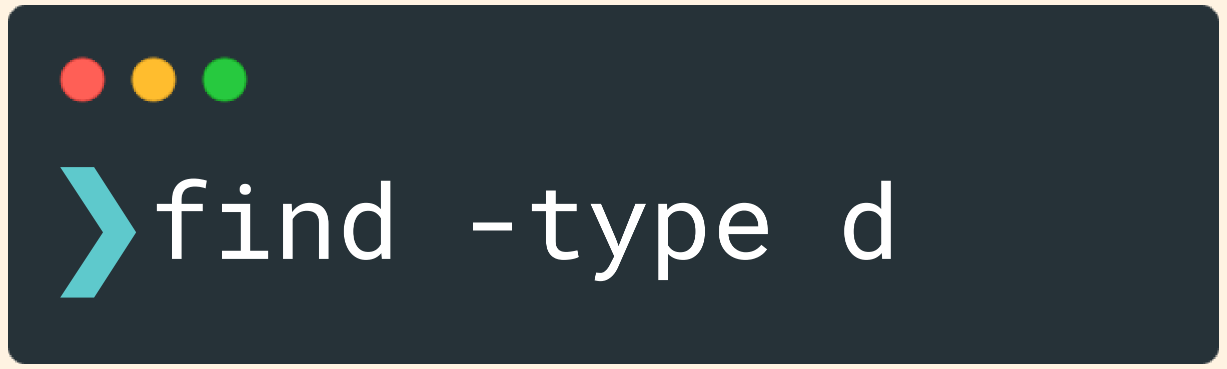
```
❯find friends/
```

finding by type

We can tell find to only find by file type: only print files, directories, symbolic links, etc using the -type option.

find -type f will limit the search to files

find -type d will limit the search to directories



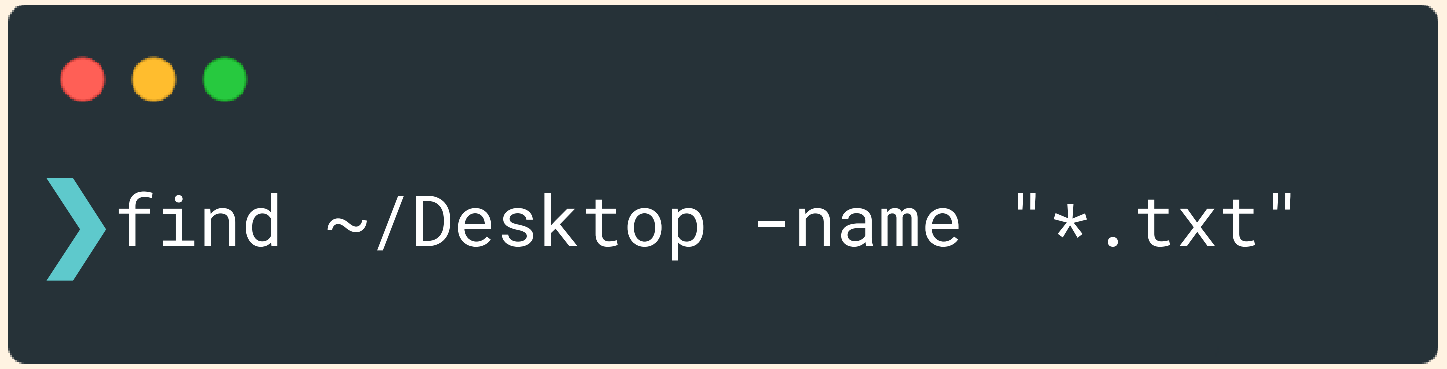
```
❯find -type d
```

finding by name

We can provide a specific pattern for **find** to use when matching filenames and directories with the **-name** option. We need to enclose our pattern in quotes.

To find all files on our Desktop that end in the .txt extension, we could run find ~/Desktop -name "*.txt"

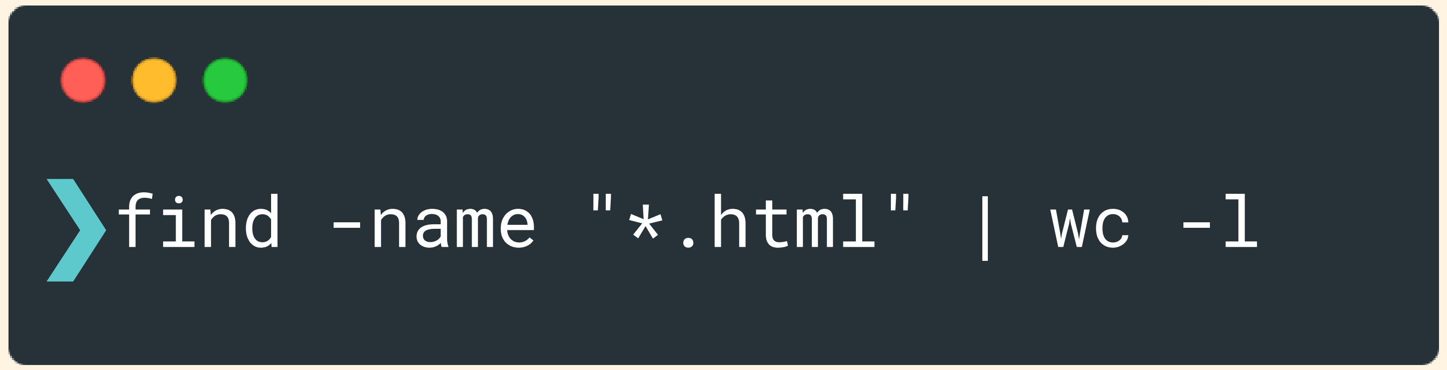
Use the -iname option for a case insensitive search



```
❯find ~/Desktop -name "*.txt"
```

Counting Results

We can pipe the output of find to other commands like word count. Use the -l option to count the number of lines (each result from find is its own line)



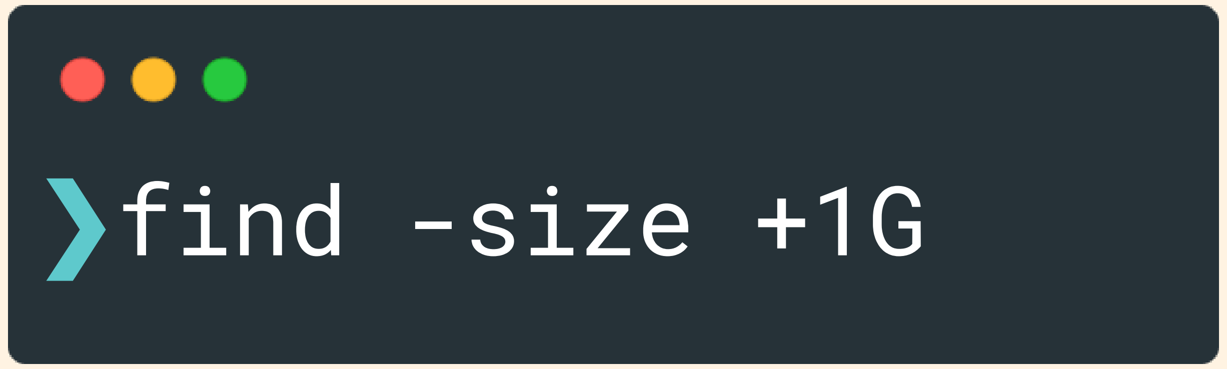
```
❯find -name "*.html" | wc -l
```

finding by size

We can use the -size option to find files of a specific size. For example, to find all files larger than 1 gigabyte we could run **find -size +1G**

To find all files under 50 megabytes, we could run find -size -50M

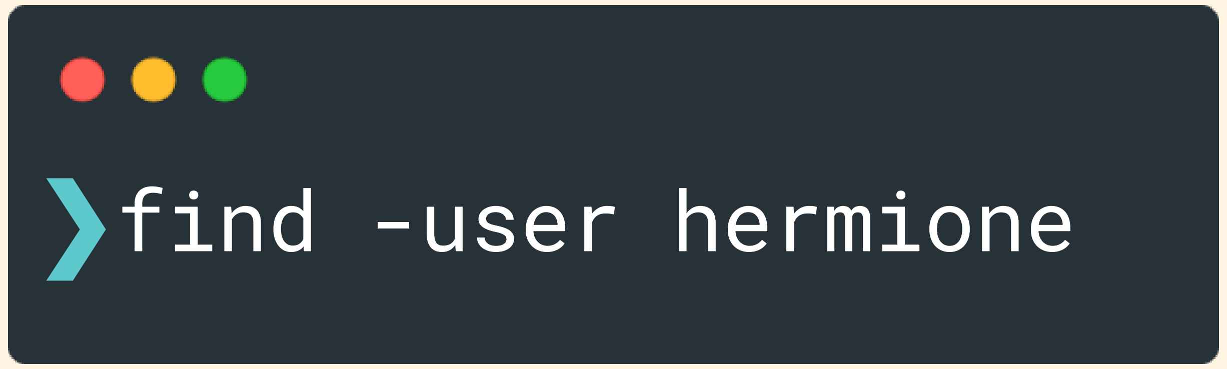
To find all files that are exactly 20 kilobytes, we could run **find -size 20k**



```
❯find -size +1G
```

finding by owner

We can use the -user option to match files and directories that belong to a particular user.



```
❯find -user hermione
```

Timestamps

mtime, or modification time, is when a file was last modified AKA when its contents last changed.

ctime, or change time, is when a file was last changed. This occurs anytime mtime changes but also when we rename a file, move it, or alter permissions.

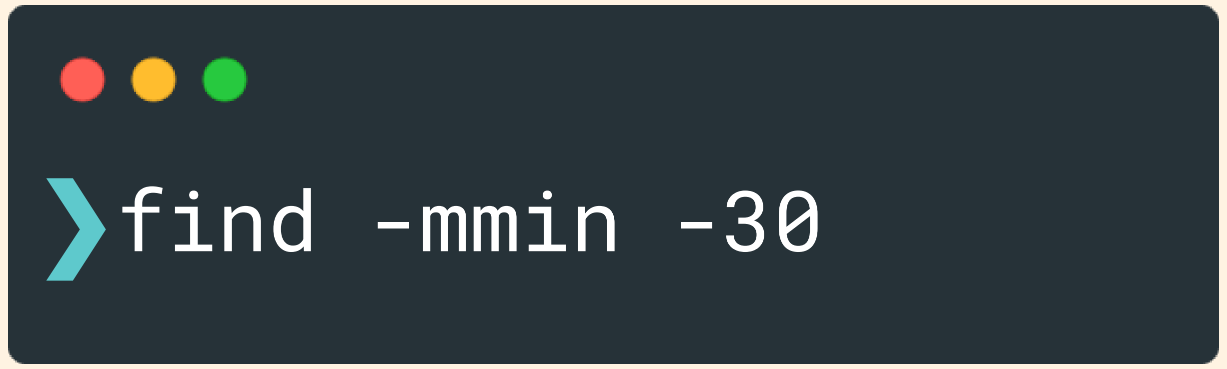
atime, or access time, is updated when a file is read by an application or a command like cat.

Finding By Time

We can use the **-mtime num** option to match files/folders that were last modified num*24 hours ago.

find -mmin -20 matches items that were modified LESS than 20 minutes ago.

find -mmin +60 matches items that were modified more than 60 minutes ago



```
❯find -mmin -30
```

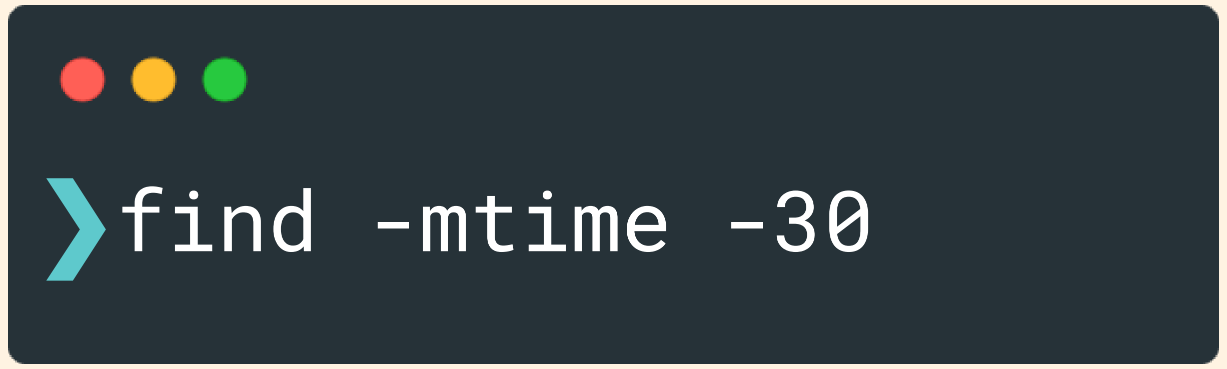
Finding By Time

-amin n will find files that were last accessed n minutes ago. We can specify +n for "greater than n minutes ago" and -n for "less than n minutes ago"

-anewer file will find files that have been accessed more recently that the provided file.

find -cmin -20 matches items that were modified LESS than 20 minutes ago.

find -cmin +60 matches items that were modified more than 60 minutes ago



```
❯find -mtime -30
```

Logical Operators

We can also use the **-and**, **-or**, and **-not** operators to create more complex queries.

```
❯find -name "*chick*" -or -name "*kitty*"
```

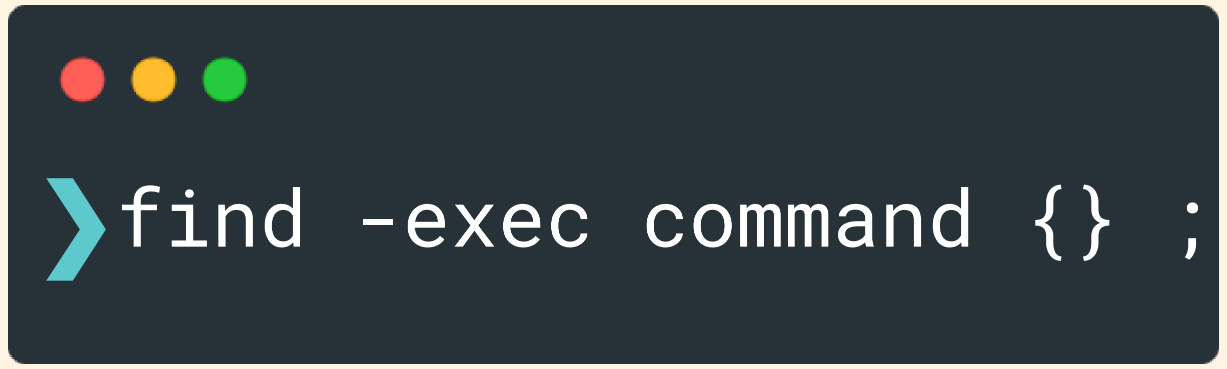
```
❯find -type -f -not -name "*.html"
```

User- Defined Actions

We can provide **find** with our own action to perform using each matching pathname.

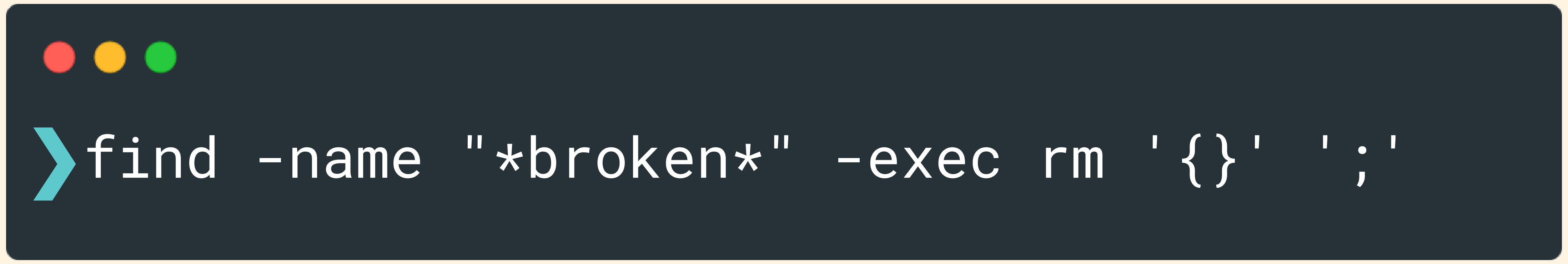
The syntax is **find -exec command** {} ;

The {} are a placeholder for the current pathname (each match), and the semicolon is required to indicate the end of the command.



```
❯find -exec command {} ;
```

User- Defined Actions



```
find -name "*broken*" -exec rm '{}' ';'
```

To delete every file that starts with contains "broken" in its file name, we could run:

```
find -name "*broken*" -exec rm '{}' ';'
```

Note that we need to wrap the {} and ; in quotes because those characters have special meanings otherwise

User- Defined Actions

```
find -type f -user colt -exec ls -l '{}' ';'
```

The above example finds all files that are owned by the user "colt", and then it lists out the full details for each match using ls -l

```
find -type f -user colt -exec ls -l '{}' ';'
```

Note that we need to wrap the {} and ; in quotes because those characters have special meanings otherwise

User- Defined Actions

```
❯find -type f -name "*.html" -exec cp '{}' '{}_COPY' ';'
```

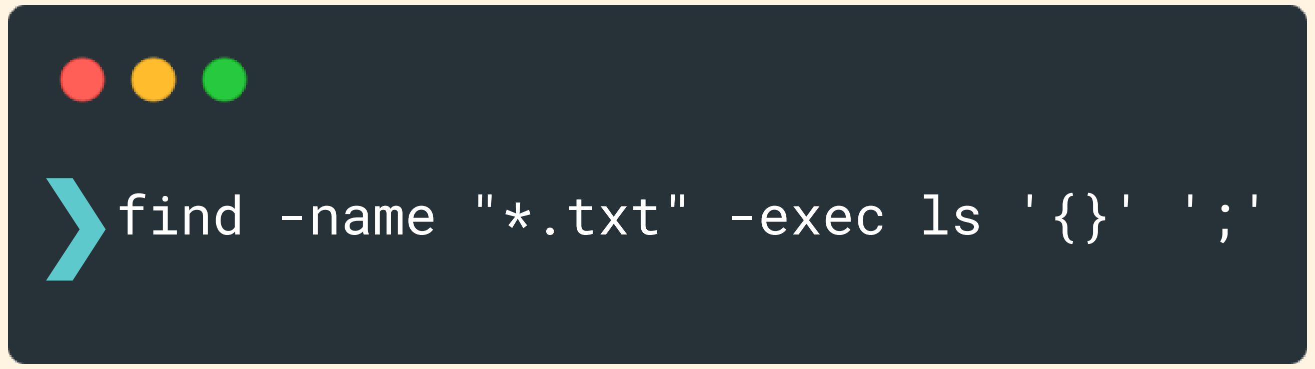
The above example finds all files that end with .html. It then creates a copy of each one using the cp command. Each of the copies ends with "_COPY" so we end up with files like "index.html_COPY" and "navbar.html_COPY"

```
find -type f -name "*.html" -exec cp '{}' '{}_COPY' ';'
```

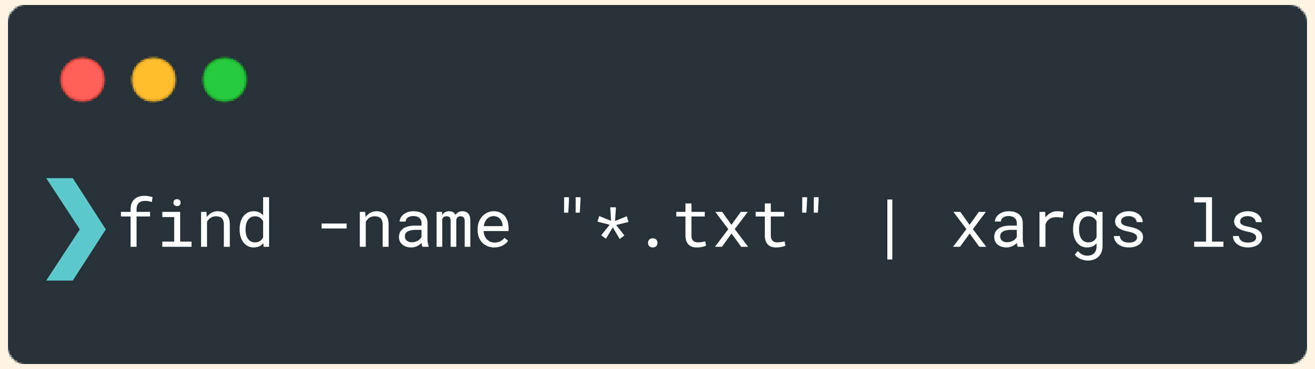
Note that we need to wrap the {} and ; in quotes because those characters have special meanings otherwise

xargs

When we provide a command via **-exec**, that command is executed separately for every single element. We can instead use a special command called **xargs** to build up the input into a bundle that will be provided as an argument list to the next command.



```
find -name "*.txt" -exec ls '{}' ';'
```



```
find -name "*.txt" | xargs ls
```

xargs

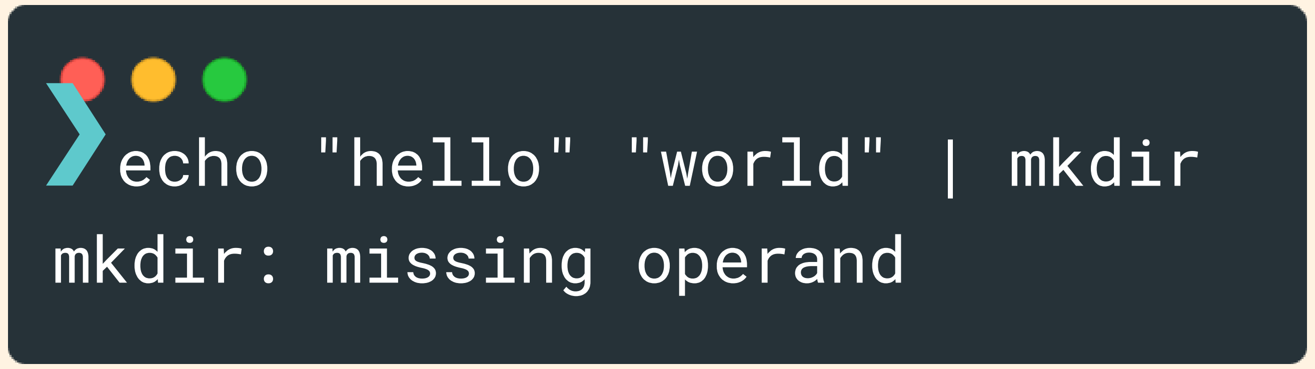
This example finds four individual chapter files (chapter1, chapter2, chapter3, and chapter4) and then passes them to the cat command, which then outputs the combined contents to a file called fullbook.txt.

```
❯find -name "chapter[1-4].txt" | xargs cat > fullbook.txt
```

xargs

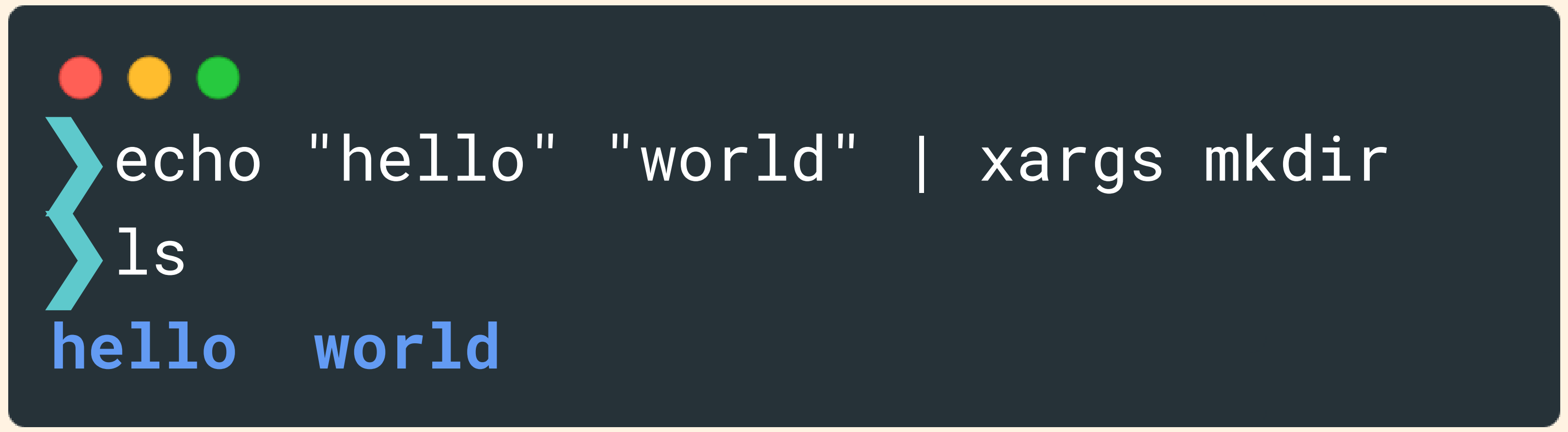
xargs reads items from standard input, separated by blanks (spaces or newlines) and then executes a command using those items

The mkdir command expects us to pass arguments. It doesn't work with standard input, so this example does NOT make any folders for us:



```
❯ echo "hello" "world" | mkdir
mkdir: missing operand
```

We can instead add in the xargs command, which will accept the standard input coming from echo and pass them as arguments to mkdir.



```
❯ echo "hello" "world" | xargs mkdir
❯ ls
hello  world
```