



```
flamealchemist final_compile
$ make clean
rm -f *.o *.out
flamealchemist final_compile
$ make clean_downloads
rm Downloads/*.*
flamealchemist final_compile
$ make
gcc -c http.c
gcc -c https.c
gcc -c common.c
gcc -o main.out main.c http.o https.o common.o -lssl -lcrypto
flamealchemist final_compile
$ ./main.out http://www.science.smith.edu/~jcardell/Courses/EGR328/Readings/KRch
5Link.pdf > log.txt
flamealchemist final_compile
$
```

Example for a http get request

Open ▾  log.txt Save ▮ - + 

```
1 protocol : http, hostname : www.science.smith.edu, dest: ~jcardell/Courses/EGR328/-
  Readings/KRch5Link.pdf
2 ip address is : 131.229.72.71
3
4 Header:
5
6 http/1.1 200 ok
7 date: mon, 05 apr 2021 11:51:18 gmt
8 server: apache/2.4.39 (ubuntu)
9 last-modified: wed, 15 feb 2012 13:29:44 gmt
10 etag: "332de-4b900b45d6200"
11 accept-ranges: bytes
12 content-length: 209630
13 connection: close
14 content-type: application/pdf
15
16 Header end
17
18 Content-Type: application/pdf
19 Content-Length: 209630
20
21
22
23 Saving data...
24
25 file has been saved!
```

Plain Text ▾ Tab Width: 8 ▾ Ln 25, Col 21 ▾ INS

1 of 9

file:///D:/Study%20Campus/PgD/cnm/lesson5.htm  
KRch5Link.pdf

95.1%

1

2

3

4

## Computer Networking and Management

### Computer Networking and Management

# Lesson 5 - The Data Link Layer

The Data Link Layer - Introduction | Link Layer and Local Area Networks | Link Layer Implementation | Error Detection and Correction Techniques | Parity Checks | Multiple Access Links and Protocols

## Lesson Outline

### The Data Link Layer - Introduction

### Link Layer and Local Area Networks

- Reliable Delivery
- Flow Control
- Error Detection
- Error Correction

### Link Layer Implementation

### Error Detection and Correction Techniques

Error Detection and Correction Scenario

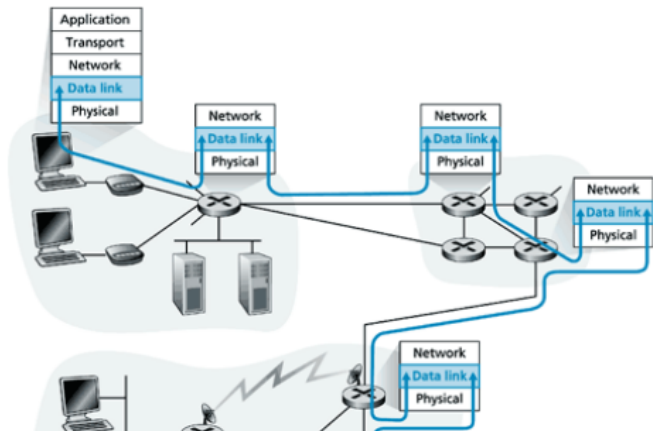
### Parity Checks

- One-Bit Even Parity
- Internet Checksum Technique
- Cyclic Redundancy Check (CRC)

### Multiple Access Links and Protocols

[GOTO TOP](#)

## The Data Link Layer - Introduction



**Goals:**

Understand principles behind data link layer services:

- error detection, correction
- sharing a broadcast channel: multiple access
- link layer addressing, reliable data transfer, flow control

Instantiation and implementation of various link layer technologies

- The 'make clean' command is used to remove the existing binary files
- The 'make clean\_downloads' command is used to remove existing downloads . (Note : Downloads are stored in the currentPath/Downloads/ folder)
- The 'make' command is used for compiling the main.c file by linking the other files and required libraries.
- The main compiling command (mentioned in the makefile) is :

```
gcc -c http.c
gcc -c https.c
gcc -c common.c
gcc -o main.out main.c http.o https.o common.o -lssl -lcrypto
```

- The same command is executed when the command 'make' is called
- For running a program the command used is :

```
./main.out <enter url>
```

- Optionally we can pipe the output to a .txt file (log.txt in this case) for which the command would be :

```
./main.out <enter url> > log.txt
```

- This is done so that the huge output doesn't confuse the user.
- The output file is stored in the Downloads/ directory'
- Some shortcuts are embedded in the makefile to check the output faster for the evaluators and the output can be checked in the 'Downloads/' folder :

```
test_http_pdf : main.out
```

```
./main.out http://www.science.smith.edu/~jcardell/Courses/EGR328/Readings/KRch5Link.pdf > log.txt
```

```
test_https_pdf : main.out
```

```
./main.out https://people.cs.umass.edu/~arun/cs453/lectures/Chapter5.pdf > log.txt
```

```
test_https_img : main.out
```

```
./main.out https://i.insider.com/602ee9ced3ad27001837f2ac > log.txt
```

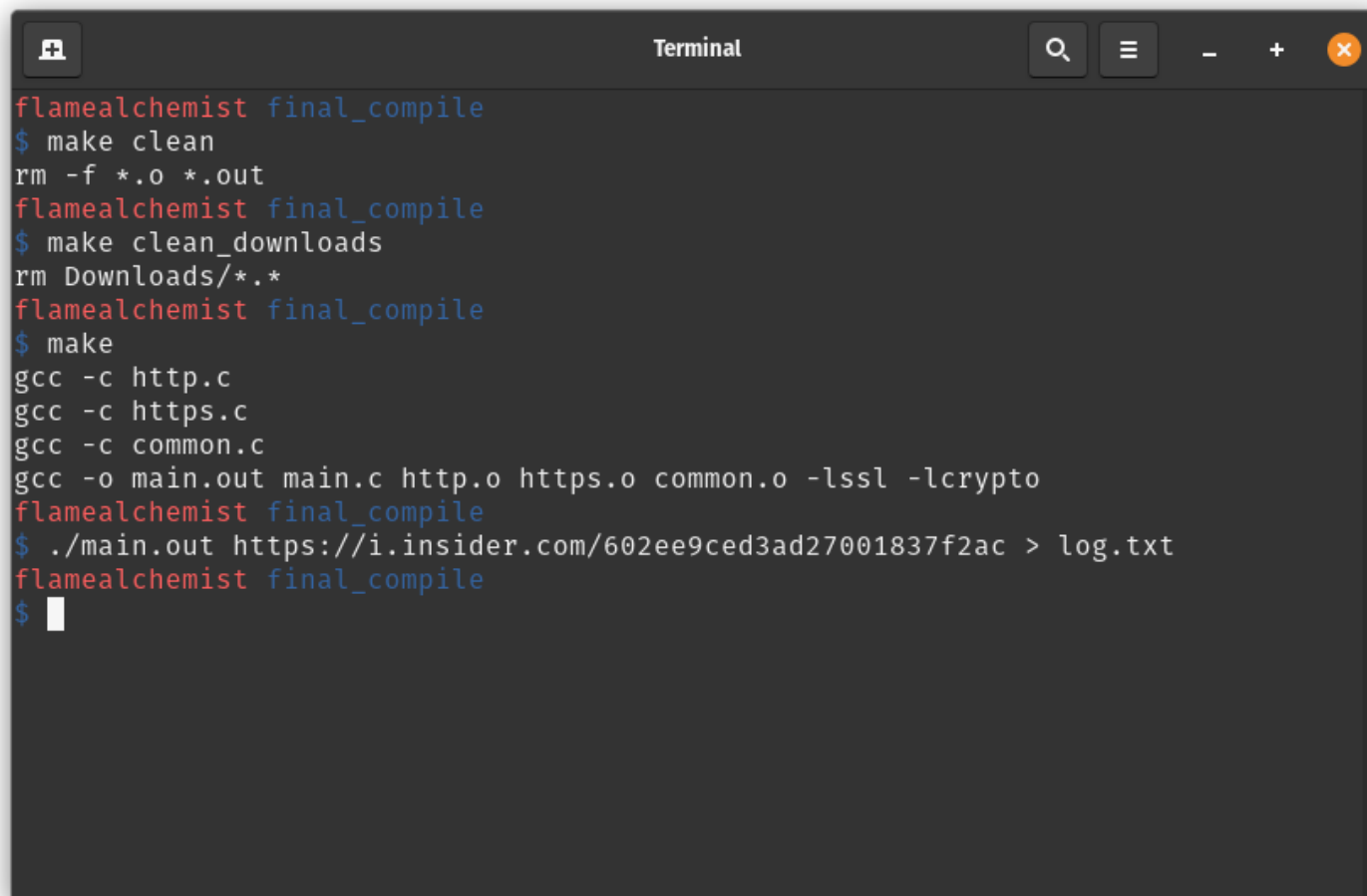
```
test_1 : main.out
```

```
./main.out https://git.20202060.xyz/dog.jpg > log.txt
```

```
test_2 : main.out
```

```
./main.out https://www.google.com > log.txt
```

Example for a https GET request:



```
flamealchemist final_compile
$ make clean
rm -f *.o *.out
flamealchemist final_compile
$ make clean_downloads
rm Downloads/*.*
flamealchemist final_compile
$ make
gcc -c http.c
gcc -c https.c
gcc -c common.c
gcc -o main.out main.c http.o https.o common.o -lssl -lcrypto
flamealchemist final_compile
$ ./main.out https://i.insider.com/602ee9ced3ad27001837f2ac > log.txt
flamealchemist final_compile
$
```

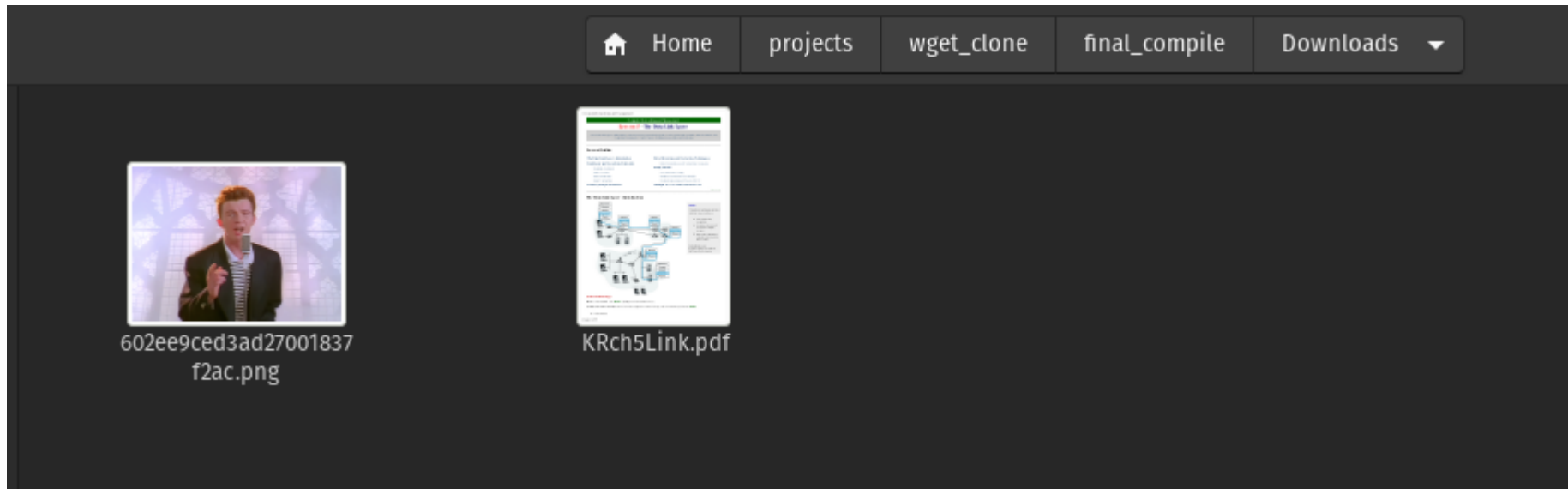
log.txt  
~/projects/wget\_clone/final\_compile

Open Save

```
1 protocol : https, hostname : i.insider.com, dest: 602ee9ced3ad27001837f2ac
2 ip address is : 199.232.254.217
3 here!
4
5
6 Header:
7
8 http/1.1 200 ok
9 connection: close
10 content-length: 1913767
11 cache-control: max-age=2592000, public
12 content-type: image/png
13 etag: "sv7e4ipmwx9k3hq76zybz0opxnumvrax3lrb3y6dzlq"
14 fastly-io-info: ifsz=1913767 idim=2343x1757 ifmt=png ofsz=1913767 odim=2343x1757 ofmt=png
15 fastly-io-warning: failed to shrink image
16 fastly-stats: io=1
17 server: amazons3
18 x-amz-id-2: et0cslxr2elc/33ns5fph98/yn33izczmjz95u/zdka89qgzxroy3qx/vklokfwfivijpldez9u8=
19 x-amz-request-id: rt2lzaqj6cdpn044
20 via: 1.1 varnish, 1.1 varnish
21 accept-ranges: bytes
22 date: mon, 05 apr 2021 12:05:28 gmt
23 age: 2454590
24 x-served-by: cache-bwi5145-bwi, cache-maa10250-maa
25 x-cache: hit, hit
26 x-cache-hits: 1, 1
27 x-timer: s1617624329.500270,vs0,ve3
28 vary: accept
29 access-control-allow-origin: *
30
31 Header end
32
33 content-type: image/png
34 content-length: 1913767
35 size of body is : 1913767
36 type of file: png
37 602ee9ced3ad27001837f2ac
38 Saving data...
39
40 file has been saved!
```

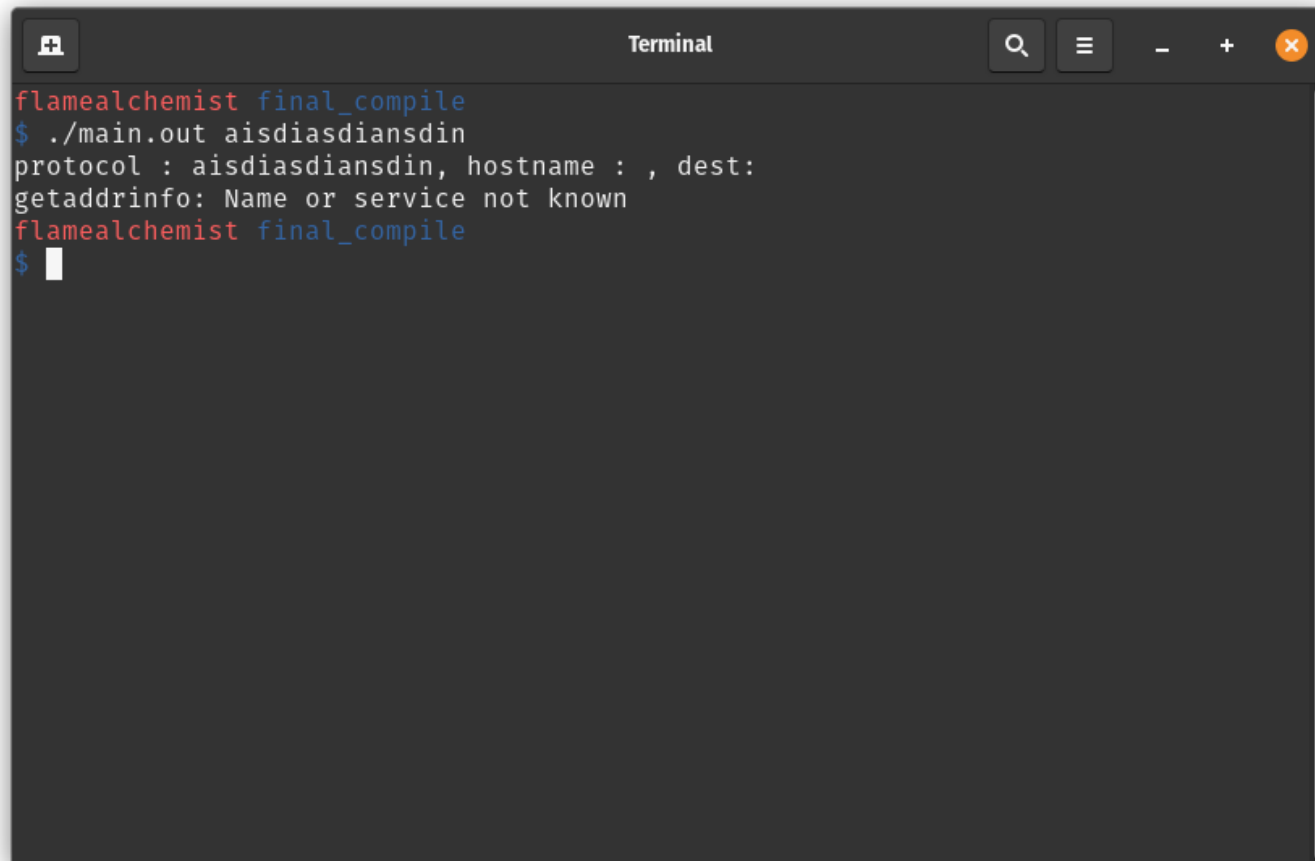
Plain Text ▾ Tab Width: 8 ▾ Ln 40, Col 21 ▾ INS





As one can observe, the file names are same as the ones provided in the URL

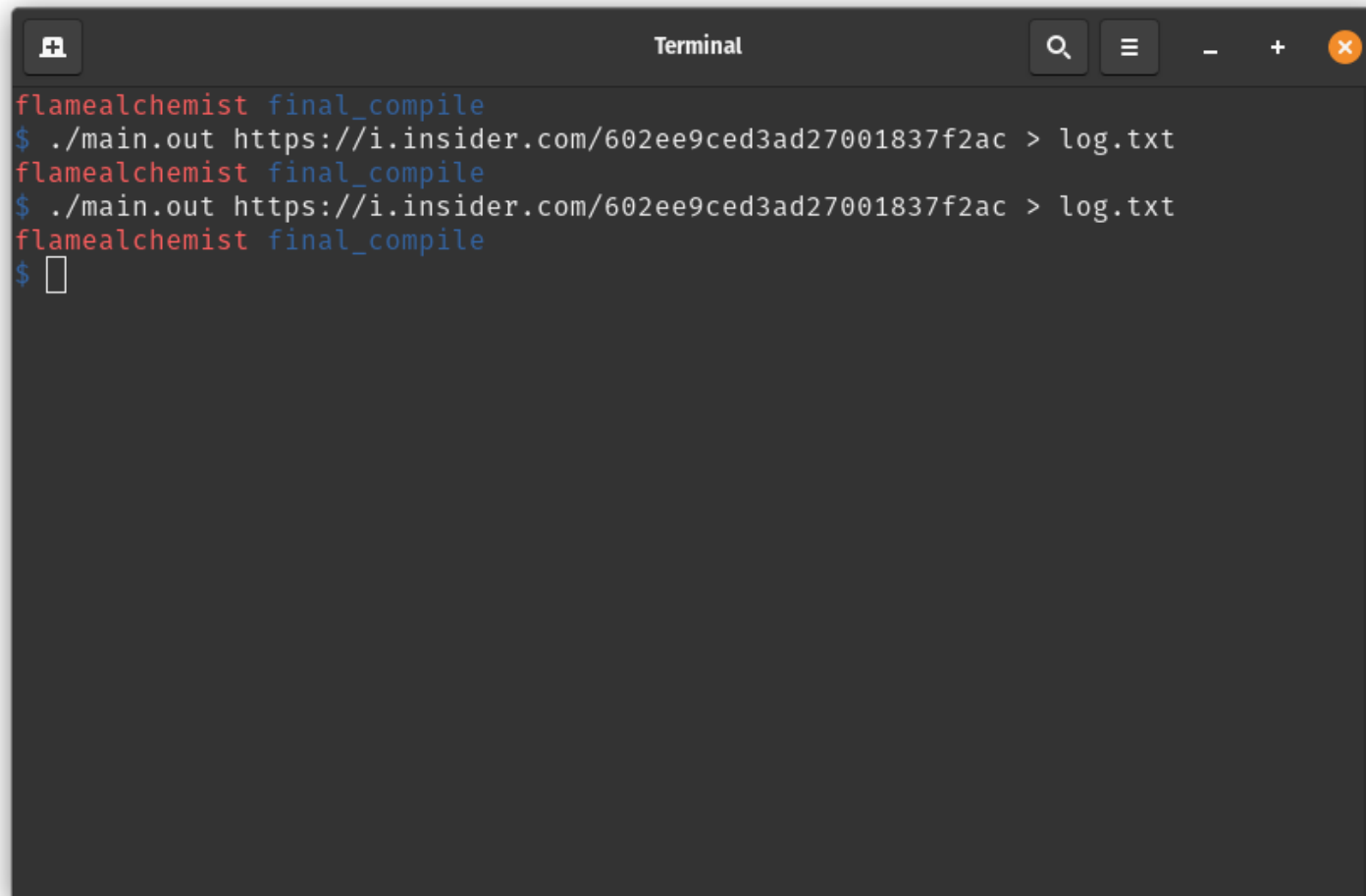


A terminal window titled "Terminal" with a dark background. The window has a title bar with a search icon, a menu icon, and window control buttons (minimize, maximize, close). The terminal content shows a user prompt, a command to run a program with an argument, and the program's output indicating a failed connection due to an unknown host. The user then enters another command.

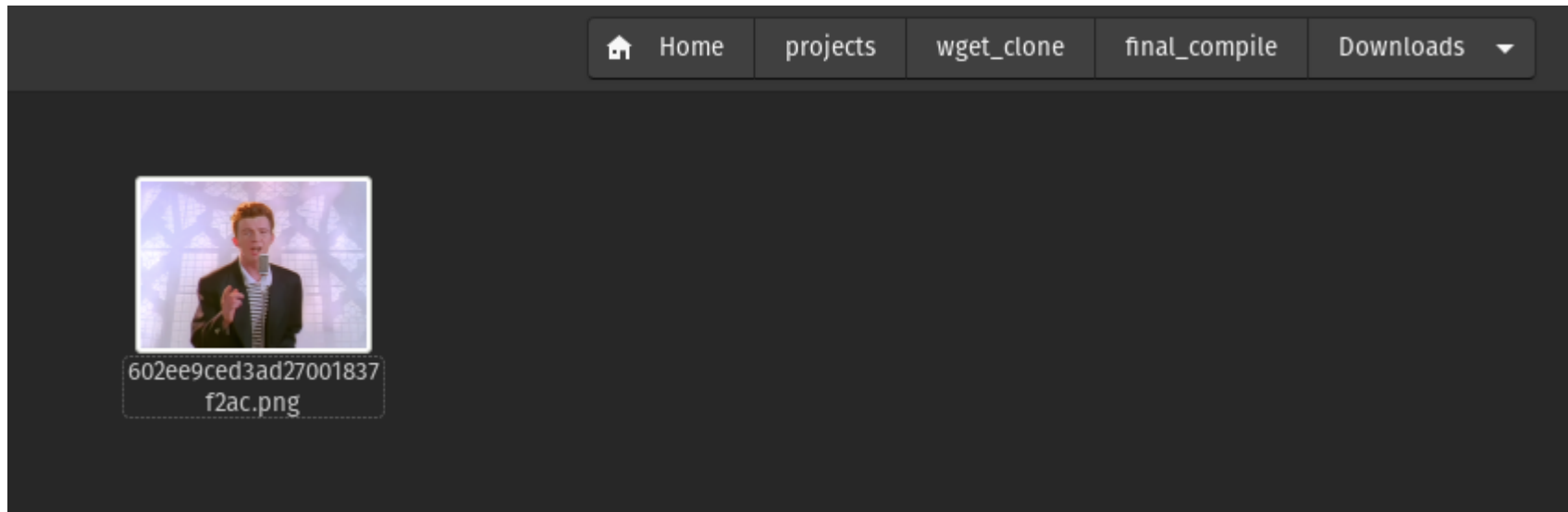
```
flamealchemist final_compile
$ ./main.out aisdiasdiansdin
protocol : aisdiasdiansdin, hostname : , dest:
getaddrinfo: Name or service not known
flamealchemist final_compile
$
```

Case when the URL is not valid

Case when same file is requested twice:

A terminal window titled "Terminal" with standard macOS window controls (search, menu, zoom, close). The terminal shows a sequence of commands and prompts. The prompt "flamealchemist" is in red, and "final\_compile" is in blue. The command prompt "\$" is in blue. The command being executed is in white. The output is in white. The terminal shows the same sequence of commands and prompts three times, with the last one ending in a cursor.

```
flamealchemist final_compile
$ ./main.out https://i.insider.com/602ee9ced3ad27001837f2ac > log.txt
flamealchemist final_compile
$ ./main.out https://i.insider.com/602ee9ced3ad27001837f2ac > log.txt
flamealchemist final_compile
$ █
```



The same file is overwritten

Sources used :

- openssl docs
- <https://stackoverflow.com/questions/62011930/c-to-perform-https-requests-with-openssl>

Nrupesh Surya U 2017B2A70767G