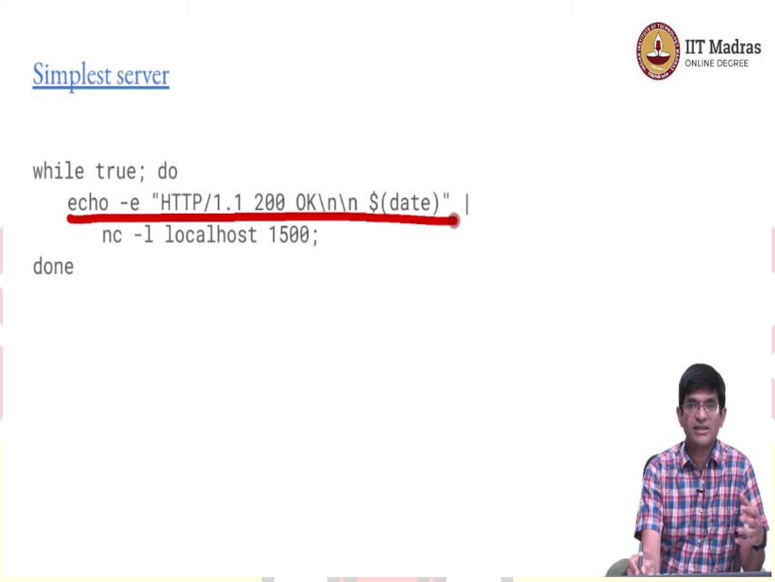
INDIAN INSTITUTE OF TECHNOLOGY MADRAS • सिद्धिर्भवति कर्मजा •

IIT Madras

ONLINE DEGREE

Modern Application Development – I
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology Madras
Simple Web Server
(Refer Slide Time: 00:16)



Simplest server

```
while true; do
    echo -e "HTTP/1.1 200 OK\n\n $(date)" |
        nc -l localhost 1500;
done
```

Hello, everyone, and welcome to this course on modern application development. So now, I am also going to start bringing in a few examples and show you a little bit about what certain kinds of code could look like. What I have shown on the screen over here is in some sense, the simplest possible web server. Not exactly the simplest possible, but close.

So, let us try and understand what is happening over here. This has been written in shell script, which means that all of these things, the while true and so on are things which you can actually type into Linux or Unix shell and they would execute as commands. And there are some specific parts of it, which I will try and explain out here and then show you later as an example, what exactly it does.

So, this initial part, the while true do, is something in shell script which basically says, keep on repeating whatever is inside or here indefinitely, until I use some other mechanism to kill this program, and the done is just sort of closing out whatever is there corresponding to the while, do. So, what really matters is, what is happening in these two lines internally. And what is happening is, it is sort of a two-stage process. You can see the symbol right at the end over here, this called the pipe symbol. So, what it says is do something with line number 1 and pipe that information into line number 2.

Those of you who are not completely familiar or comfortable with Linux or Unix-based systems I would strongly recommend that you do try out and get some familiarity at least with the basics of scripting. And the reason being that even though you do not absolutely need it in order to write a web app, for example, it helps a lot to understand what is happening behind the scenes.

For the most part, whatever is required for this course, I will be explaining as we move along, but in case there are parts that are not very clear, you might have to look outside for further information about them. So, now let us take these two lines and understand them individually. The first line is fairly simple. All that it says is echo, and echo is a very simple Unix command. Echo minus e in this case, just basically, whatever is there within this quote, symbols over here the double quotes, it just prints it out.

Normally, it would print it out onto the screen. But the way that Unix works is that it actually prints it out onto what is called the standard output, STD out. And STD out can be redirected using this pipe symbol, so that instead of going onto a screen, it now goes into the second program. So, what exactly is the echo outputting? It is printing out some information over here.

The first part of it is what you can see HTTP slash 1.1. So, remember, I am creating a server. This is the output of the server, which means this is the information that goes back to the client. And HTTP slash 1.1 basically says, this server is responding with an http 1.1. code. What is the code 200 OK. So, HTTP code number 200, HTTP defines something called status codes.

200 is one of the status codes, 301 is another one, which indicates that a page has moved. Probably one of the ones that you know you are most familiar with is the 404 the 404 code, which basically says page not found. In fact, it is got to the point where it has almost become the subject of meanness. So, something is gone 404 basically means it is missing.

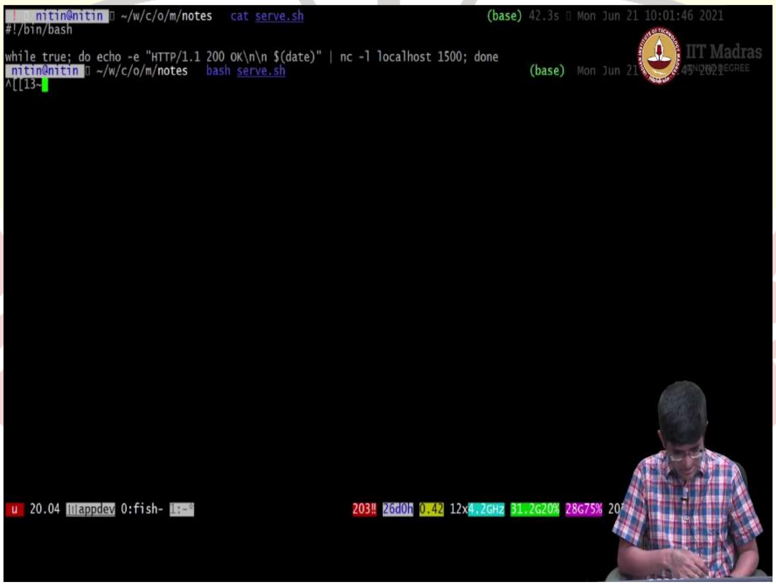
But where did that come from? It is an HTTP status code. The server responds with 200 if it says everything is OK. 404 if something what is not found. Now, in this particular case, as you can see, this server is always going to say 200 OK, because I just have a while loop where all that it does is print out 200 OK, this backslash and backslash n is to leave two lines, blank lines, that is how the HTTP header and data are separated. And what is the information that it gives out after all that? The date. So, the dollar and within parentheses date is a way of telling the shell script to actually run this command date and take whatever information comes from there in place of that dollar parenthesis.

So, this echo command in other words is going to print out the present date. But is it going to print it onto the screen? No, it is going to pipe it into this second command out here, which is nc. This is a command called netcat, and netcat is going to listen on localhost that is the local server, port number 1500.

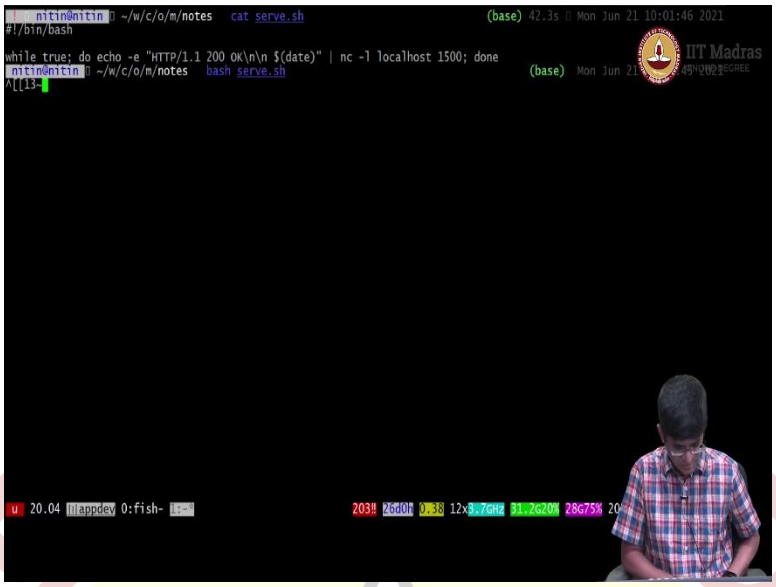
So, that is it. That is what this server is now doing. Essentially, what it is saying is the nc minus l localhost 1500 is the part that is actually implementing the server. What that means is nc will cause the operating system to open a port and will listen. And any time a request comes in on port number 1500, that will be fed to nc.

What does nc do? It will just take whatever came from the echo and send it out to whoever asks anything on port 1500. So, the nc is not really doing anything very bright. It does not interpret the request. All the receivers request came on port 1500 fine, respond with this. The point is, this is sufficient to create a very trivial form of web server. Let us just see what this looks like in practice.

(Refer Slide Time: 06:31)



```
nitin@nitin  ~/w/c/o/m/notes   cat serve.sh                    (base) 42.3s  Mon Jun 21 10:01:46 2021
#!/bin/bash

while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l localhost 1500; done
nitin@nitin  ~/w/c/o/m/notes   bash serve.sh                         (base)  Mon Jun 21 
^[[13~

u 20.04 appdev 0:fish- 1:-*                    203‼ 26d0h 0.42 12x4.2GHz 31.2G20% 28G75% 20
```



```
 nitin@nitin  ~/w/c/o/m/notes   cat serve.sh                    (base) 42.3s  Mon Jun 21 10:01:46 2021
#!/bin/bash

while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l localhost 1500; done
 nitin@nitin  ~/w/c/o/m/notes   bash serve.sh                          (base)  Mon Jun 21
^[[13~
```

So, what I have done is, I have connected to a Linux-based machine. And what you can see is that this program that I just indicated before, the while true, do the echo, pipe and nc and the done I just put everything together on one line. Just to show that you can create a web server in one line of code.

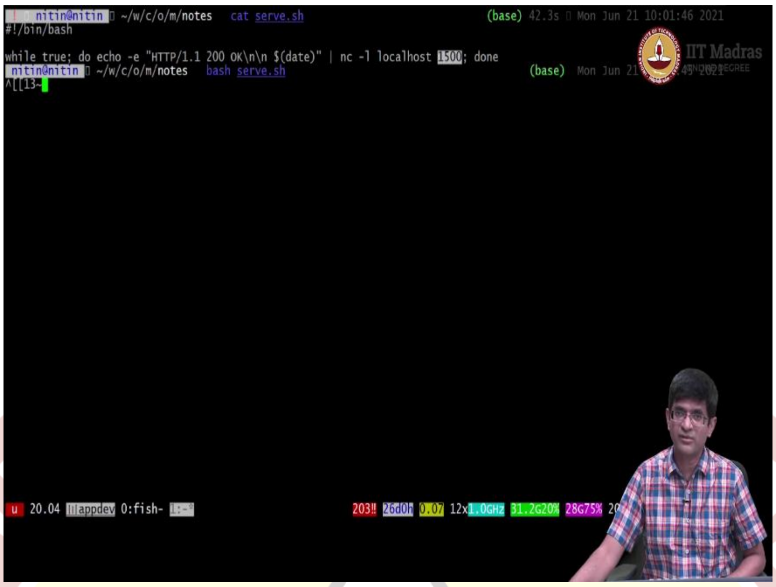
What is this part out here right at the top bin slash bash that is just something that we add at the beginning of shell scripts to indicate what should be the server that is to be used for this. Not strictly speaking necessary, in this case, it is just a hint to the operating system, so that it becomes a bit easier for the operating system to know and run this code.

So, now if I take this, and I actually run it by typing bash, by the way, is the name given to what is called the Bourne Again Shell. It is one of the shells. A shell is something that we use in order to interact with the Unix-based system, so bash is the shell that we are using for this particular set of scripts.

So, I have this small script. I know that what it is supposed to do is run a while loop, which is just going to go into an infinite loop and wait for information. So, when I run this, nothing happens. And then when I think about it, that makes sense because all that it is doing is it is basically doing a while, true echoing something, piping it through nc minus l and waiting. It does not really have anything else to do over here.

So, how do we check what, whether this is going to work or not? for that? Now, I know that this is actually working like a web server, can I connect to it in some way? and retrieve that information and see does this make sense or is it behaving the way that I wanted, what I am going to do is I will open another shell for that.

(Refer Slide Time: 08:43)



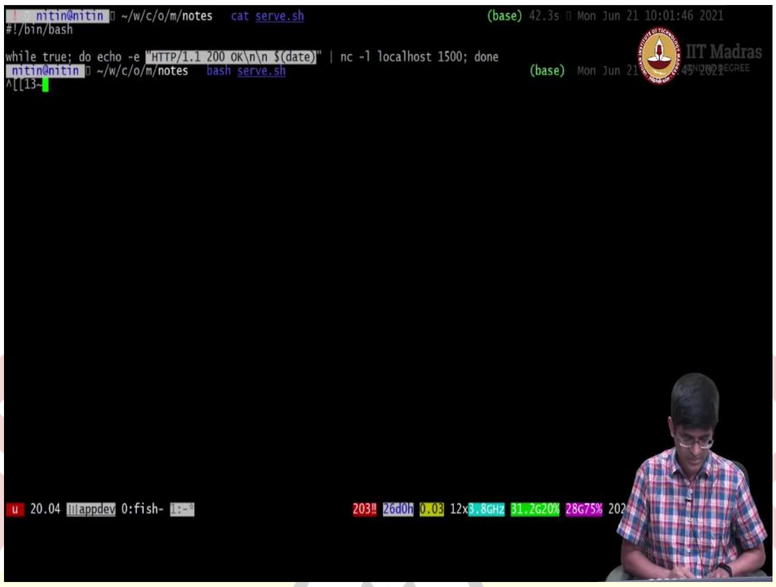
```
nitin@nitin  ~/w/c/o/m/notes   cat serve.sh                    (base) 42.3s  Mon Jun 21 10:01:46 2021
#!/bin/bash

while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l localhost 1500; done
 nitin@nitin  ~/w/c/o/m/notes   bash serve.sh                          (base)  Mon Jun 21
^[[13~
```

And what I have in this other shell over here is, that I will use a command called curl. Curl is, again a very useful command, in general, for people who are doing app development. And the reason is it allows you to create custom requests to a web server. It takes care of a lot of the background thing, like sending the right kind of headers, and so on all of that information is taken care of by curl, so we can do it without really having to worry about the details of the HTTP protocol.

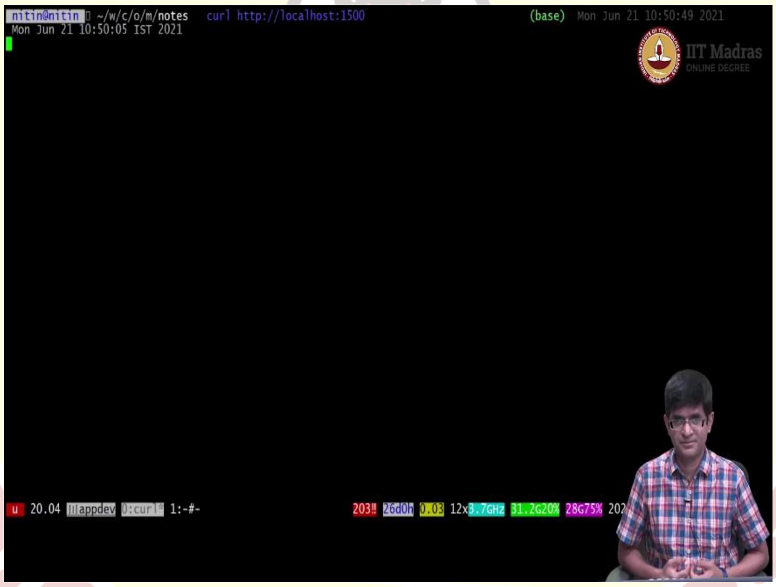
Now, in this case, what I know is that I can send a request. What kind of request? It should be an HTTP request, the colon slash, slash is how I sort of separate out the protocol, the HTTP protocol, from the rest of the thing that is the server and the path on the server. Everything put together is what is called the URL the Uniform Resource Locator. In this case, the HTTP says that the server that I am connecting to speaks the HTTP protocol.

What that means is, it expects HTTP requests and will give me a response which is a valid HTTP response. Now, go ahead and connect. Now, which is the server? I know that it is on the same machine because all I did was open another shell over here. So, I am going to type in localhost. But one thing to keep in mind is, you remember what we saw earlier, this shell, this script is connecting on port number 1500, not on the regular HTTP port.

(Refer Slide Time: 10:26)



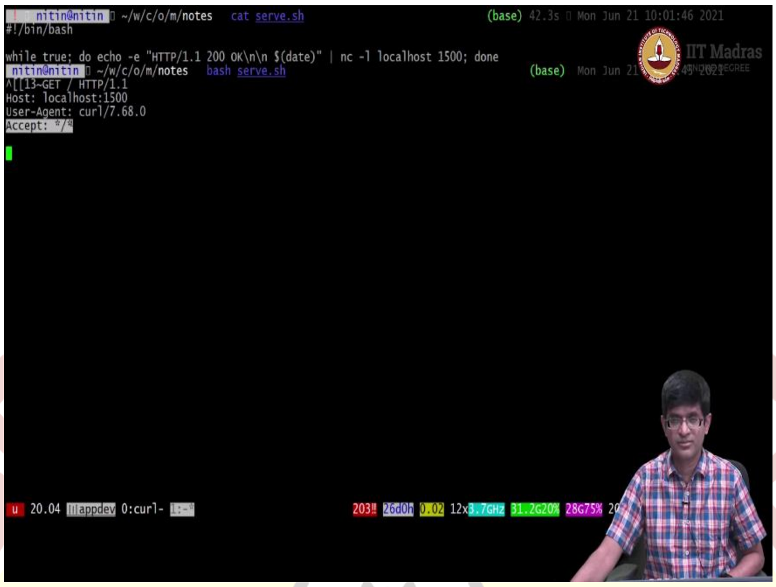
```
nitin@nitin  ~/w/c/o/m/notes   cat serve.sh                    (base) 42.3s  Mon Jun 21 10:01:46 2021
#!/bin/bash

while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l localhost 1500; done
nitin@nitin  ~/w/c/o/m/notes   bash serve.sh                          (base)  Mon Jun 21
^[[13~

u 20.04 appdev 0:fish- 1:-                 203‼ 26d0h 0.03 12x3.8GHz 31.2G20% 28G75% 202
```



```
nitin@nitin  ~/w/c/o/m/notes   curl http://localhost:1500             (base)  Mon Jun 21 10:50:49 2021
 Mon Jun 21 10:50:05 IST 2021

u 20.04 appdev 0:curl 1:-#-                203‼ 26d0h 0.03 12x3.7GHz 31.2G20% 28G75% 202
```

So, let me put that information in there. I just say localhost colon 1500. And now when I do this, can we think about what we expect to see? What was my server doing is just basically going to print out this information and then print the date, which means that what I am going to get back from the server will be all of this, this header information and the date. But remember that curl is actually a full HTTP client, which means that it would not show you the header information. What it will show you is just the date.

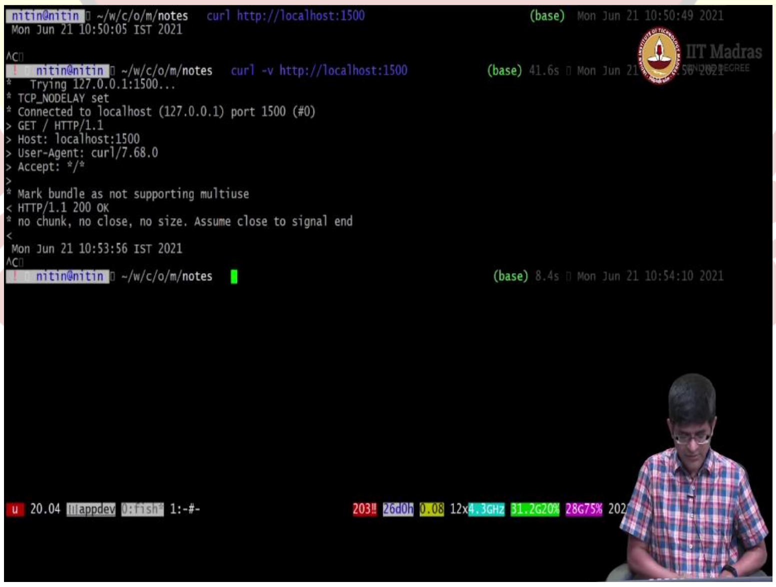
(Refer Slide Time: 11:14)



```
nitin@nitin  ~/w/c/o/m/notes   cat serve.sh                (base) 42.3s  Mon Jun 21 10:01:46 2021
#!/bin/bash

while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l localhost 1500; done
nitin@nitin  ~/w/c/o/m/notes   bash serve.sh                     (base)  Mon Jun 21
^[[13~GET / HTTP/1.1
Host: localhost:1500
User-Agent: curl/7.68.0
Accept: */*
```

If I go back over here, I now see that this there was a get request. And this request over here get slash HTTP slash 1.1 was what came from curl. It also said that it wanted to connect to the host, localhost colon 1500, it also specified what kind of user agent it is, gave some information about itself, and said, what is it willing to accept, which is pretty much everything.

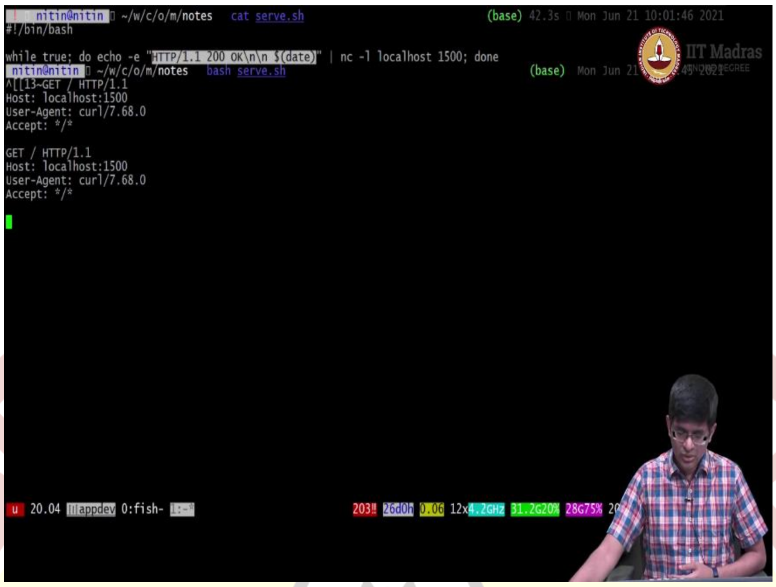
(Refer Slide Time: 11:44)



```
nitin@nitin  ~/w/c/o/m/notes   curl http://localhost:1500          (base)  Mon Jun 21 10:50:49 2021
 Mon Jun 21 10:50:05 IST 2021

^C
nitin@nitin  ~/w/c/o/m/notes   curl -v http://localhost:1500    (base) 41.6s  Mon Jun 21
*   Trying 127.0.0.1:1500...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 1500 (#0)
> GET / HTTP/1.1
> Host: localhost:1500
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
* no chunk, no close, no size. Assume close to signal end
<
 Mon Jun 21 10:53:56 IST 2021
^C
nitin@nitin  ~/w/c/o/m/notes                                    (base) 8.4s  Mon Jun 21 10:54:10 2021
```

The response came as this piece of information. If I run that with a verbose flag, that is a dash v over here, I will find that I get a bit more information, and we will look at this in a little bit more detail.

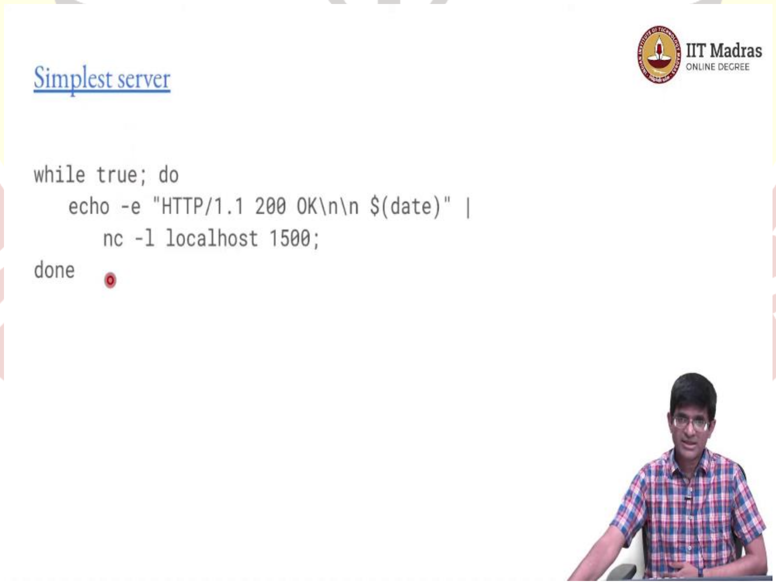
(Refer Slide Time: 12:05)



```
nitin@nitin  ~/w/c/o/m/notes   cat serve.sh                    (base) 42.3s  Mon Jun 21 10:01:46 2021
#!/bin/bash

while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l localhost 1500; done
 nitin@nitin  ~/w/c/o/m/notes   bash serve.sh                         (base)  Mon Jun 21
^[[13~GET / HTTP/1.1
Host: localhost:1500
User-Agent: curl/7.68.0
Accept: */*

GET / HTTP/1.1
Host: localhost:1500
User-Agent: curl/7.68.0
Accept: */*
```

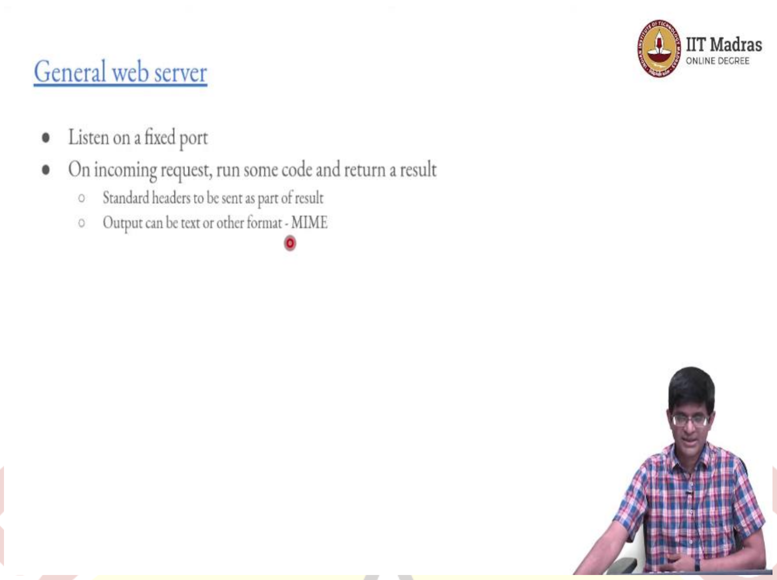
So, in other words, what you can see from this is that and once we go back to the server, we can see that, I got one more request. One more piece of information from curl, all of this, these four lines essentially came from curl, and all I did was respond with this, which showed up as the date on the other end. So, let us look at this in a little bit more detail.

(Refer Slide Time: 12:29)



Simplest server

```
while true; do
   echo -e "HTTP/1.1 200 OK\n\n $(date)" |
      nc -l localhost 1500;
done
```



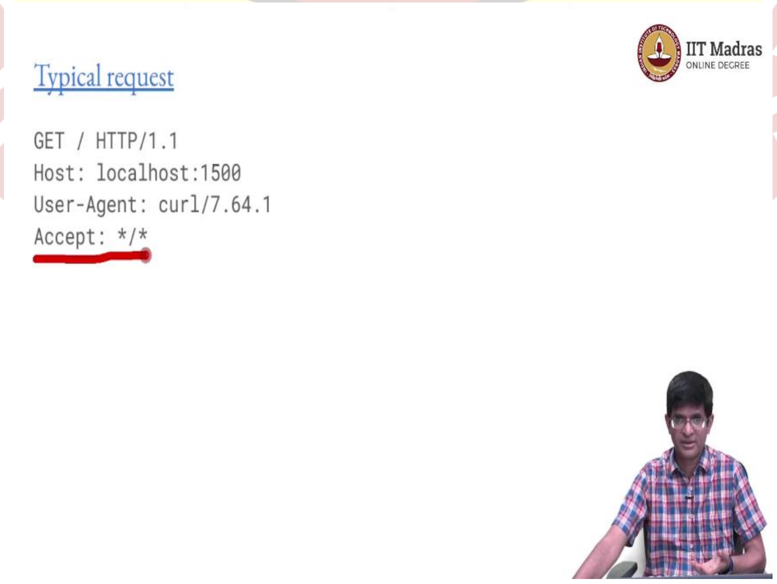
General web server

- Listen on a fixed port
- On incoming request, run some code and return a result
  - Standard headers to be sent as part of result
  - Output can be text or other format - MIME

So, in other words, what we have is, the server itself just had these four commands. And all that was doing was listening on a fixed port, 1500. Whenever the incoming request came, it ran some code, the echo and the dollar date, created a result actually on the fly, so in this sense, this is a fancy web server, it is doing something dynamic.

But it has standard headers that are to be sent as part of the result, and the output can be either text or some other format, which is usually called a mime format. MIME stands for, I think, multimedia Internet Mail Extensions, it was defined for the purposes of sending email, but in email you could have different kinds of attachments. So, that is where MIME originally came into the picture, but it is been extensively used in different parts of the web since then.

(Refer Slide Time: 13:24)



Typical request

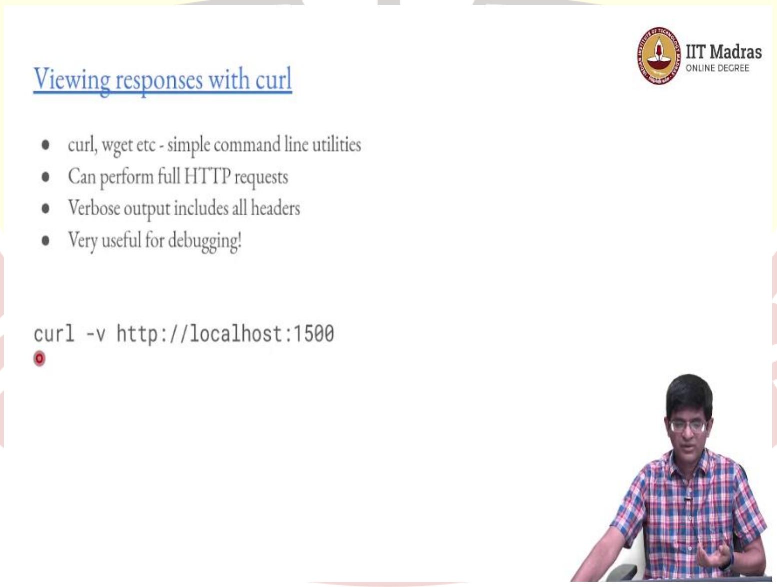
```
GET / HTTP/1.1
Host: localhost:1500
User-Agent: curl/7.64.1
Accept: */*
```

So, as we saw right here, this is what a typical request looks like. This is what went from curl to the web server. This part, the get HTTP slash 1.1 is a standard thing. It has to be there, that has to be that is the actual request. The next three lines are actually optional. But in the case of HTTP 1.1, you at least need to specify this host because HTTP 1.1 says that I could have multiple hosts running on the same machine, so I need to specify which one I am interested in talking to.

The user agent is optional, but usually useful. Most web servers need to see the user agent so that they can decide how to respond. If they see that it's a command line interface like this, they might just give a very trivial response. If they see that you are running Chrome, they may respond in a different way, if they see that you are running Firefox a different response, not always required or even a good idea, but possibly. Most of them require this thing.

And finally, there is Accept star slash star. The star slash star is a way of indicating the MIME types that the client is willing to accept. In this case, it is curl, all that is going to do is take whatever it gets and just dump it out onto the screen, which is why it is saying star slash star. I will accept anything. So, this is what a request looks like.

(Refer Slide Time: 14:54)



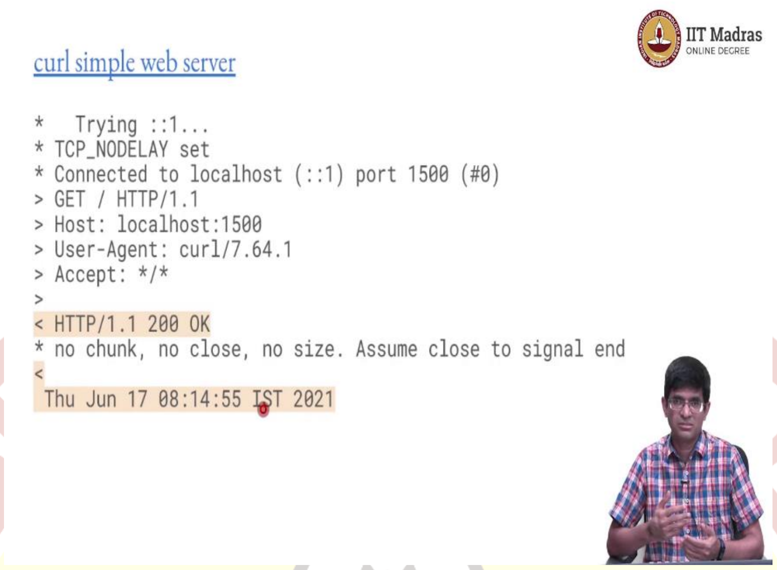
Viewing responses with curl

- curl, wget etc - simple command line utilities
- Can perform full HTTP requests
- Verbose output includes all headers
- Very useful for debugging!

```
curl -v http://localhost:1500
```

And this is pretty much what we did. We just run the curl minus v HTTP colon slash, slash localhost colon 1500.

(Refer Slide Time: 15:05)


curl simple web server

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 1500 (#0)
> GET / HTTP/1.1
> Host: localhost:1500
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
* no chunk, no close, no size. Assume close to signal end
<
 Thu Jun 17 08:14:55 IST 2021
```

This is the full response. And let us analyze it in a little bit more detail because it helps to understand the sort of back and forth the protocol between the client and the web server. So, what we have is, this everything starting with a star over here, these three lines as well as this line down here are just debug information from curl, they are not part of the protocol.

It is not sent from the client to the server or back, it is just printed out for you, as debugging information. So, as you can see, what it says is it is trying colon, colon one. This colon, colon one, by the way, is an IP address. But unlike the format that I mentioned earlier, which is just four numbers separated by dots, this is actually something called an ipv6 the version 6 address.

For now, you can just ignore it colon, colon one means localhost means your own machine. This TCP no delay set is, again, some part of the TCP tuning settings that are done over there. It I believe, it controls how fast the connection will be set up, and how quickly it will try to ramp up the speed at which it is trying to download information, but for now, you can just ignore it.

So, the good news is the third line, which says it connected to localhost. If there was no server running on localhost, it would have said, either access denied or server did not respond or something like that, but in this case it is good. It connected to localhost Port 1500. And after that this last part, ignore it. We do not really need to worry too much about what is this information over here, it is just pretty much curl saying, I reached the end, I did not get any further information.

What is useful is to instead look at what was sent from curl to the server, and it is these actually these five lines. Look at this last greater than over here, this blank line is important. Unless you give the blank line or there, the server does not know that you are done. So, curl basically sends out a GET slash HTTP slash 1.1. So, GET is the request. It is the type of request, it is basically saying, I want to get some information from you. What information? Slash. Slash is a way of indicating that I just want information your basic core information, the route information.

I am not giving you any further details on what I want, just give me the whatever is your starting point. So, to say. And it is also mentioning that the protocol it is using is HTTP slash 1.1. There are many different variants of this. There is HTTP slash 1.0, which is almost not used these days. 1.1 is still the predominant one, the most commonly used version of the protocol, HTTP 2.0. has nothing whatsoever to do with web 2.0. HTTP 2.0 is a different variant of the protocol that allows certain other things.

So, for example, it allows pipelined connections and so on, it is used in quite a few places these days, especially for things which require continuous connectivity and like downloading large amounts of data, and in fact, HTTP 3 is also in the works. It has not yet widely deployed, but it exists. But for all practical purposes, the most dominant version of the protocol, even today is HTTP 1.1.

All it matters is, you are giving this information about the host, the user agent and what you are willing to accept and then you give a blind client to say, okay I am done. Nothing more from my side, I am waiting for your response. Immediately, the response comes back, from the server. The first thing it says is HTTP 1.1 200 OK. Remember where this came from? This is exactly from the shell script, the echo part, HTTP slash 1.1 200 OK. That was just hard coded into our server.

It will always respond with HTTP slash 1.1 200 OK, because that is all that there is in the code. Even if you sort of make a mistake in what you type into the curl, as long as it connects on this port, this is the response you will get, and then there are two backslash ns. Remember, backslash n, backslash n. One is to say go to next line, the next is, leave one more blank line after that.

Just like in the case of the incoming requests, those two blank lines or rather the one blank line over there, indicates end of headers start off information. And the information itself corresponds to the date. So, it basically just responds with the date that it has over here. So, ultimately, that is what is happening. I mean, all that the client is doing is sending a request in pretty much text format, so you can actually read the requests and make sense of what is being sent. The server also sends back its headers in text format, followed by the actual data that it wants to respond to. So, this in other words is the full sort of back and forth in a simple web communication.