



Linux for Researchers

Chapter 12: The World Wide Web

This is a huge subject that's very relevant to many of us these days. Because it's so large, I've had to pick out just a few topics to talk about. I hope the things I've chosen will be useful for you.

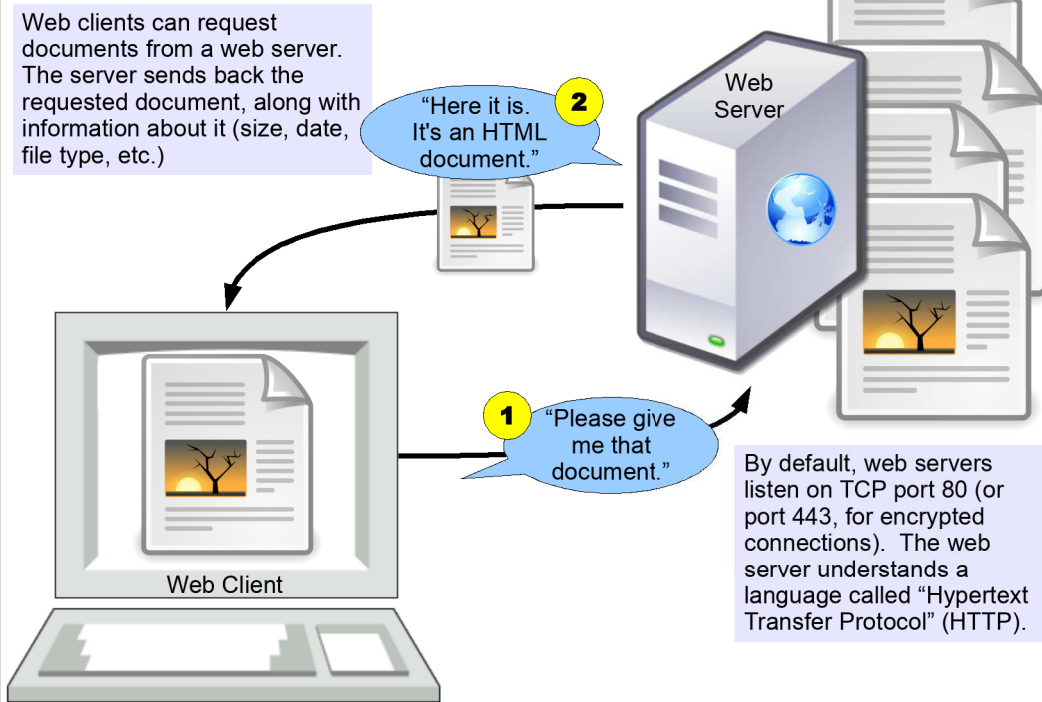
Part 1: What is the Web?



Let's start out by defining terms, beginning with “web servers”.

Note that the term “web server” is commonly used in two different ways. It might refer to an actual, physical computer, or it might refer to the software inside that computer that actually does the work. I hope it'll be clear from context which one of these I mean.

What's a Web Server?:

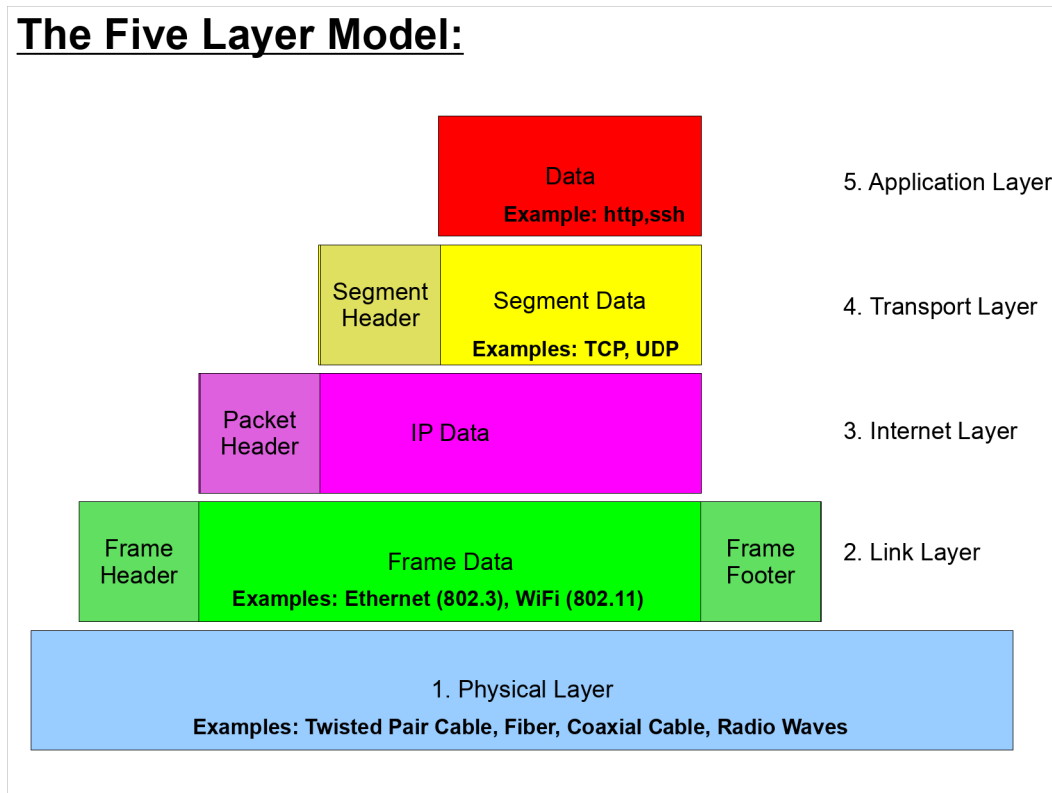


A web server is a computer that's capable of serving up documents through the "Hypertext Transfer Protocol" (HTTP). A network service running on these computers listens to TCP port 80 and/or port 443, and responds to HTTP requests.

When the web server returns a document, it also returns some metadata about the document: its size, creation time, and file type. The file type is expressed as one of the "Multimedia Internet Mail Extention" (MIME) types. Some common types are "text/html", "application/pdf", and "text/plain". The file type helps the client decide how to deal with the file.

Interactive web clients are usually referred to as "web browsers".

The Five Layer Model:



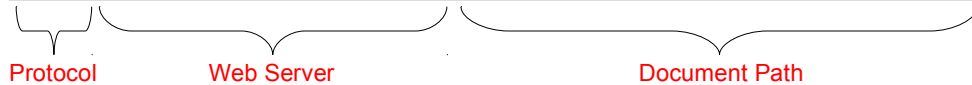
Http is an example of the 5th layer of our five-layer network model. It sits at the “application layer”, on top of the physical layer (cables, radio waves...), the link layer (ethernet, usually), the internet layer (which lets us connect to the internet) and the transport layer (which gives us access to port numbers, among other things).

The ssh protocol is another example of something that works in the application layer.

Uniform Resource Locators (URLs):

Each document on the web is identified by a unique address called a “Uniform Resource Locator” (URL). Here's an example:

```
http://myserver.example.com/products/swatters/flyswatters.html
```



The “protocol” will be either “**http:**” or “**https:**” for documents on web servers. URLs beginning with other protocols refer to other types of server (“**ftp:**”, for example). Web browsers are usually capable of talking to more than just web servers.

If the URL begins with “**https:**” the web client will use an encrypted (“secure”) version of http to fetch the document from the server.

The most general form of a URL is:

```
protocol://host:port/documentpath?querystring#anchor
```

The “**querystring**” can be used to send information to the web server (e.g., search terms) and the “**anchor**” can be used to refer to a specific location within the given document.

If we gave the first URL to a web client, we'd be telling the client: “Connect to a web server running on port 80 on `myserver.example.com`, and ask the server to give you `'/products/swatters/flyswatter.html'`.”

Default Documents:

In the following URL, the document path ends with the file name “flyswatters.html”:

`http://myserver.example.com/products/swatters/flyswatters.html`

If you specify a URL without a file name, like:

`http://myserver.example.com/products/swatters`

the web server will decide, on its own, what to do. Usually, the server will look in the specified directory for a default document with a name like “**index.html**”, “**default.html**”, “**home.html**”, “**welcome.html**”, or something similar, as defined in the configuration options for the particular web server.

For example, the web server might convert the URL above into:

`http://myserver.example.com/products/swatters/index.html`

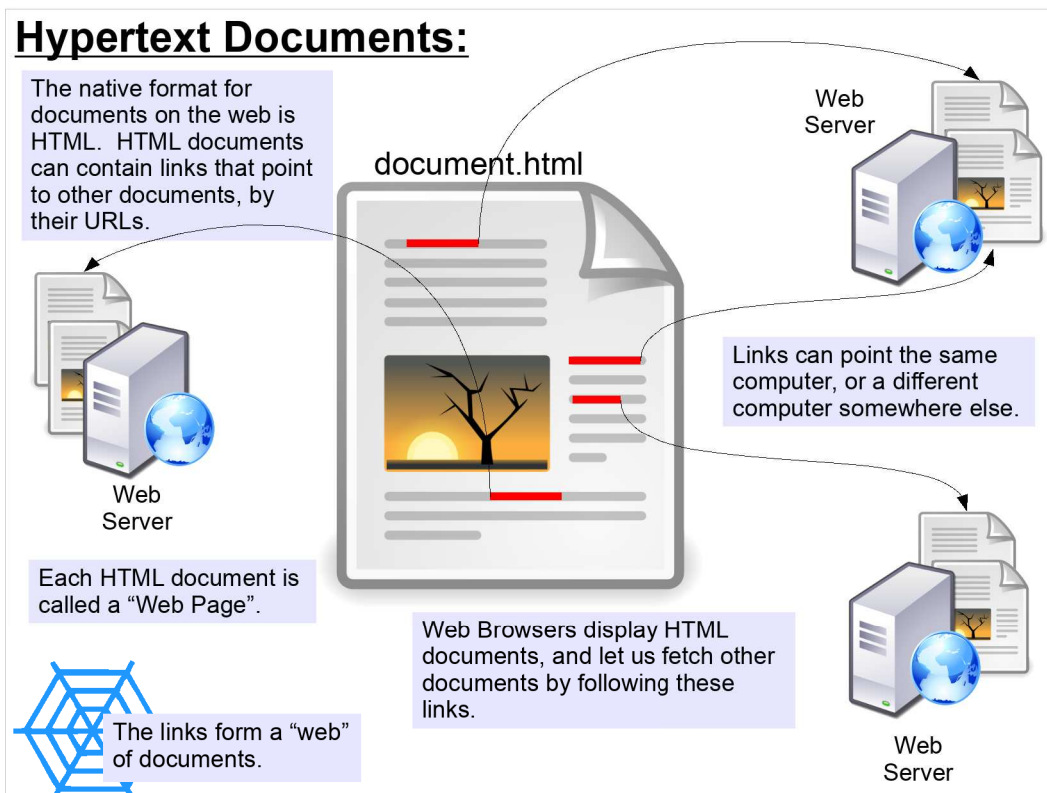
If only the server name is given, like this:

`http://myserver.example.com`

It might be converted to this:

`http://myserver.example.com/index.html`

If a given directory doesn't contain a default document, like “index.html”, the server may alternatively choose to return an automatically-generated index of all of the files in the given directory. The format of this index, and whether or not the server is willing to generate it, will depend on the server configuration.



Even though web servers can serve up any type of document, the native format for documents on the web is "hypertext". Hypertext documents are just text documents that have been "marked up" (annotated) to indicate:

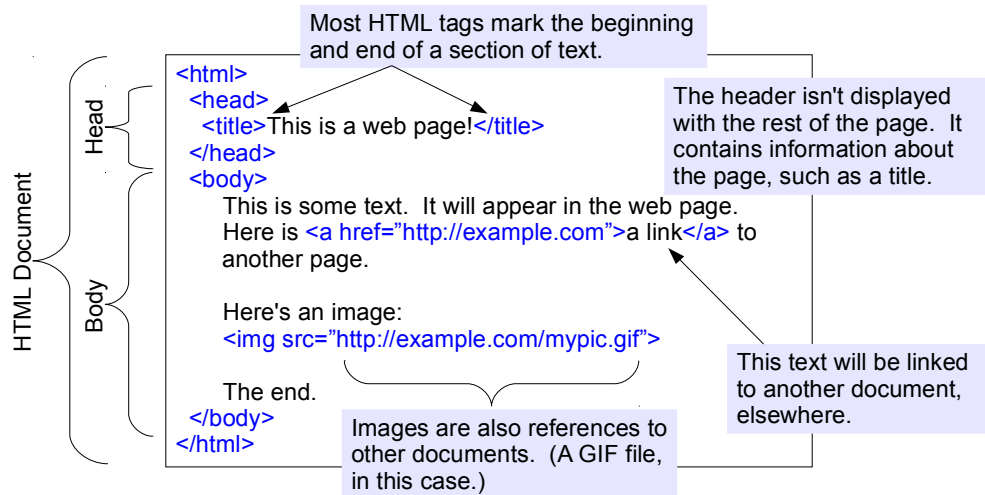
- font and other layout preferences,
 - document structure (e.g., paragraphs, sections),
 - references to other documents,
- etc.

This markup is done using "Hypertext Markup Language" (HTML).

The references (called "links") to other documents are particularly important. These links allow you to jump directly to related documents, and from there to other documents, following a "web" of references.

HTML Syntax:

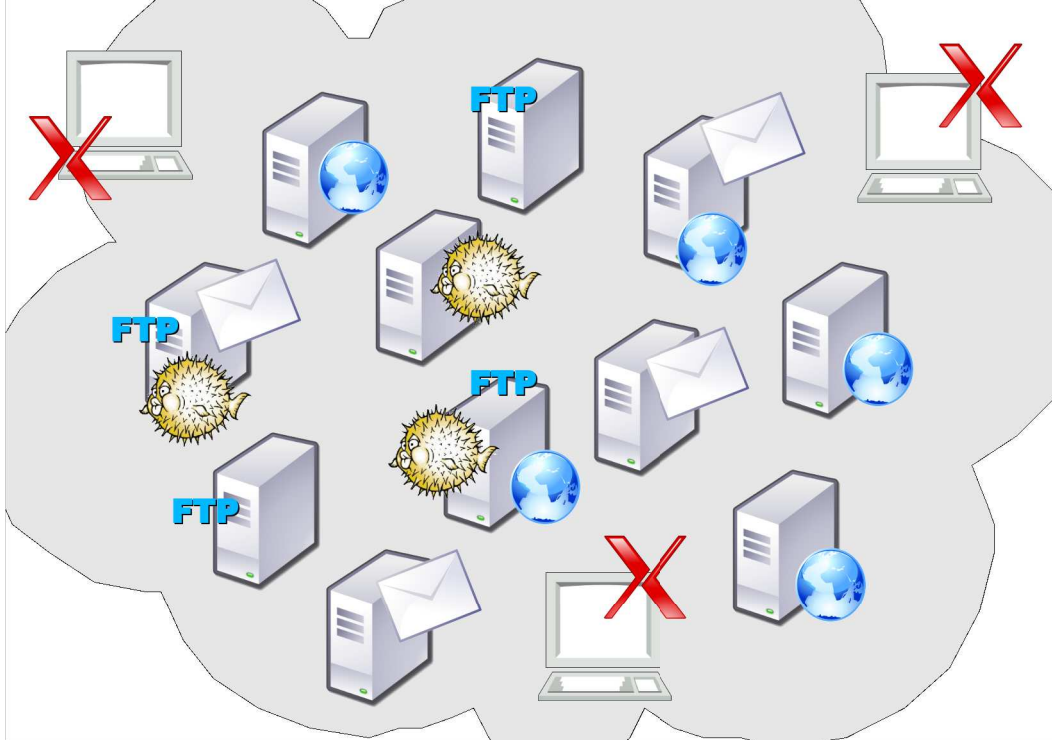
HTML markup is done by adding tags to the text in a document. Each tag consists of directives enclosed in angle brackets (“<>”).



For a complete list of HTML tags and other information, see:
<http://www.w3.org/TR/html5/>

This isn't a course in HTML, but here's a simple example showing what an HTML document looks like, in case you're not familiar with it.

The “Internet” versus the “Web”:



Since the web is so important to us now, it's sometimes conflated with the internet that underlies it. There are many types of computers on the internet. Only some of them are web servers. There are also mail servers, ssh servers, ftp servers, X servers, and many others. Some computers have several of these roles. Some computers don't provide services at all, and only act as clients.

I'll give you three definitions of “the web”. Here's Definition #1: “The web is the set of all HTTP servers on the internet”.

Alternatively, here's Definition #2: “The web is the set of interlinked documents served up by HTTP servers.”

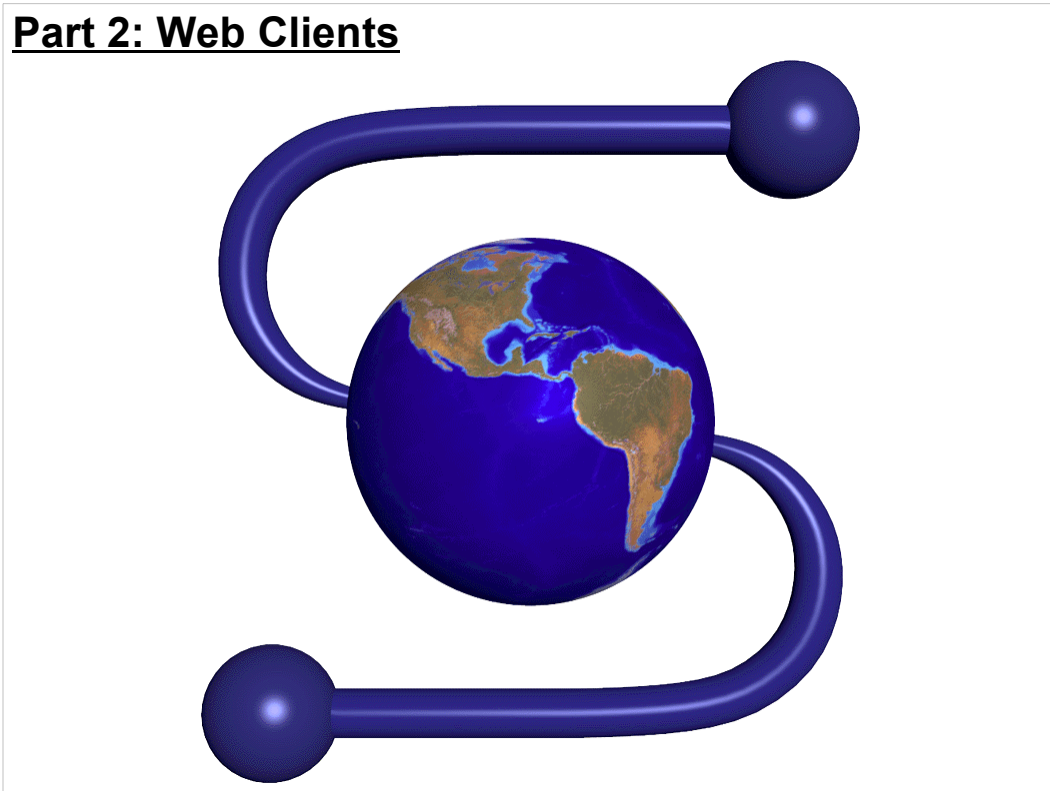
Finally, the broadest, Definition #3: “The web is the set of documents accessible through a web browser.”

What Web Servers Do:

- **Deliver documents** on demand.
- Let you specify which document by giving a slash-separated **document path**.
- Can deliver documents in **any format** (HTML, PDF, Word, Excel, plain text, etc.).
- Listen on TCP **port 80** (by default), or **443** (by default) for encrypted connections.
- Understand the **HTTP** protocol (e.g., GET and POST).

These are necessary and sufficient requirements for a web server.

Part 2: Web Clients



This is the symbol for Mosaic, the web browser developed in 1993 by the National Center for Supercomputing Applications (NCSA). I think you could make a good argument that Mosaic was one of the most important pieces of software ever written.

It was web browsers that made the web popular, not web servers. Mosaic gave the average person easy access to resources on the internet for the first time.

The “wget” Command:

“wget” is a command-line web client. It's very useful for fetching documents from the web:

```
~/demo> wget http://example.com/products/catalog.pdf
--08:47:33-- http://example.com/products/catalog.pdf
Resolving example.com... 192.168.7.3
Connecting to example.com|192.168.7.3|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1009277 (986K) [application/pdf]
Saving to: `catalog.pdf'

100%[=====>] 1,009,277 1.41M/s in 0.7s

08:47:34 (1.41 MB/s)-`catalog.pdf' saved
[1009277/1009277]
```

We usually think of graphical, interactive web clients (web browsers), but here's a text-based, non-interactive web client.

An alternative to wget is “curl”. If you don't have wget, try this instead. If neither is available, you can fetch a text version of a single web page with “links -dump http://example.com” or “lynx -dump http://example.com”.

More “wget” Examples:

The following command would fetch the document, along with any associated documents that are necessary to display the page properly (images, for example):

```
~/demo> wget -p http://example.com/products/list.html
```

Suppose we want to get all of the PDF files linked to from a particular web page. Here's how to do it:

```
~/demo> wget -r -l1 --no-parent -A "*.pdf" \  
http://example.com
```

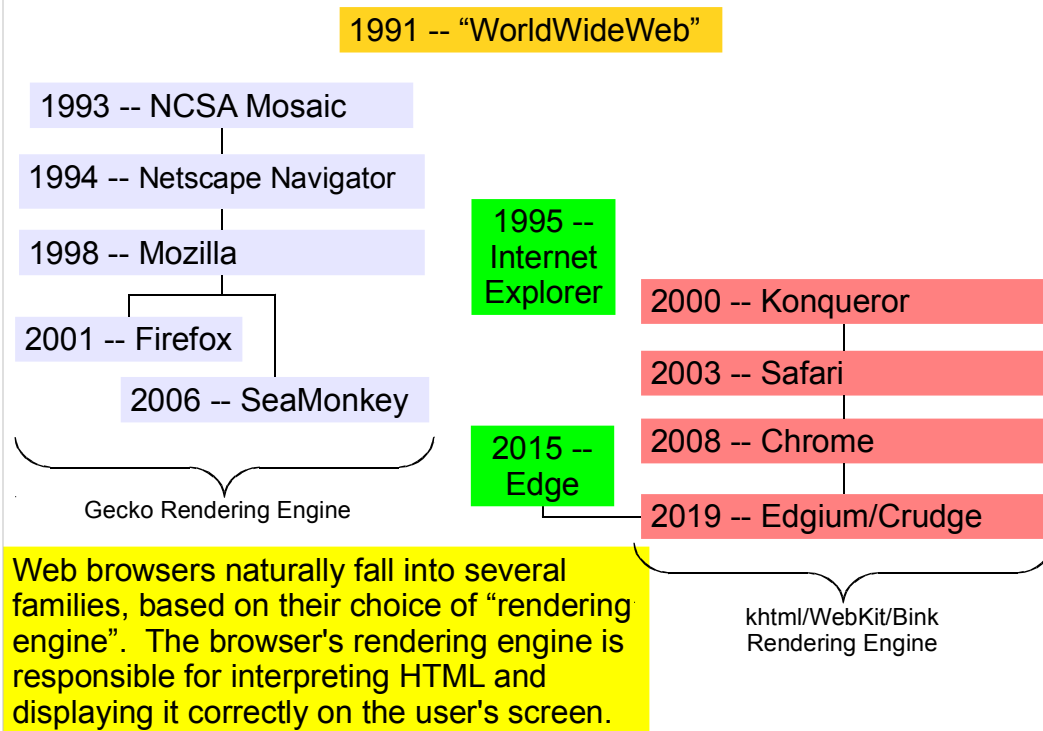
wget can also create a mirror of an entire web site:

```
~/demo> wget --mirror http://example.com
```

With “--mirror”, wget will start with the specified URL, follow all links from it to other URLs on the same server, and repeat the process until it has downloaded all the pages it can find on that server. Please take care with this. See “man wget” for more information.

For mirroring, also see “httrack”: <http://www.httrack.com/>

Interactive Web Clients (“Browsers”):



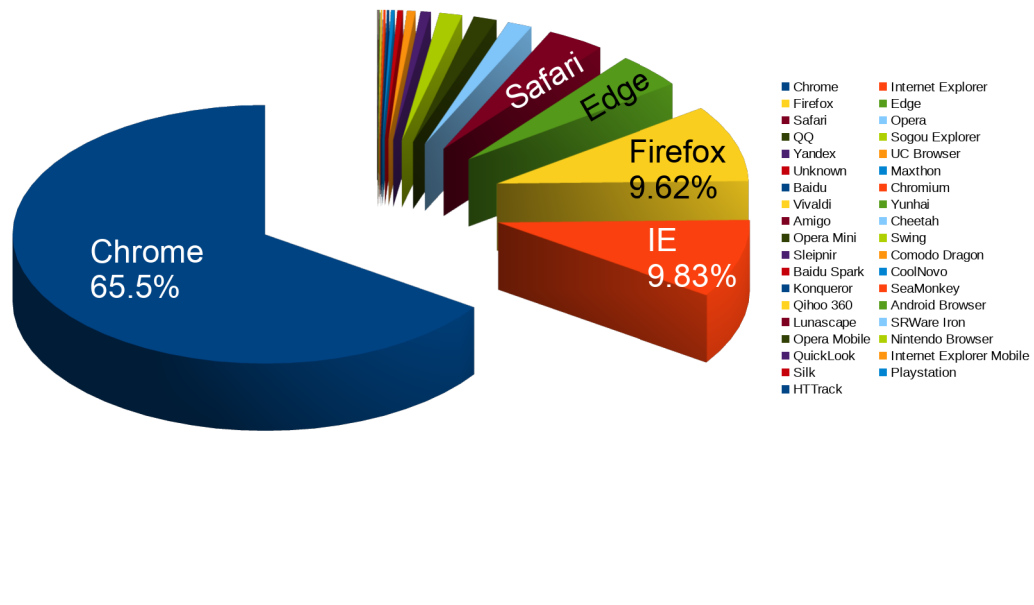
It would be hard to overstate the importance of NCSA Mosaic. It was the first popular web browser. It was freely available for many platforms, most importantly Microsoft Windows. Mosaic gave users a new way to explore the resources on the internet. Many of these resources had been available for a long time (ftp archives, usenet newsgroups), but accessing them required some skill and effort. Mosaic suddenly put all of these resources just a click away.

Web browsers are almost solely responsible for the “web revolution” of the 1990s. Without them, all of the other web-related technologies (web servers, the HTTP protocol, the HTML file format) would have been irrelevant to the average person.

A note on the name “Mozilla”, coined by Jamie Zawinski: While developing Netscape Navigator, he gave it the internal name “Mosaic Killer”, shortened to “Mozilla”.

Popular Web Browsers:

- Browser share from netmarketshare.com as of March, 2019:



Although different sources see different distributions of browsers, most agree that Chrome is currently the most widely-used web browser.

What Web Browsers Do:

All web browsers can do the following:

- **Interpret URLs**, to extract the protocol, host and path.
- **Connect to a web server** and request a document.
- **Display HTML documents**, and provide the user with a mechanism for easily following links.

Most web browsers can also do these things:

- Fetch documents from **FTP** servers, or other types of server.
- Understand "**mailto:**" URLs, and invoke a local mail program to handle them.
- Maintain a list of **external viewers** for various MIME types (e.g., PDF, Word, Excel) and invoke the appropriate viewer for non-HTML documents.
- Allow for **plug-in extensions** to handle some types of delivered content within the browser itself (e.g., Flash, Java)
- Save a **list** of selected URLs ("bookmarks" or "favorites").
- Maintain a **cache** of recently-fetched documents.
- Maintain a repository of "cookies".

In the past, most web browsers included a wysiwyg editor for HTML. This was even true for the first graphical browser, created by Tim Berners-Lee. Then, it was commonly believed that most people browsing the web would also be creating documents on the web. As it turned out, the vast majority of users only read things from the web. Web documents are created by a relative few.

Of course, this is changing as social networking sites like Facebook lead more and more people to indirectly create documents on the web. These documents don't require the user to create actual HTML documents, though. None of the currently most popular web browsers (Firefox, Internet Explorer and Chrome) includes an HTML editor.

Part 3: Web Servers

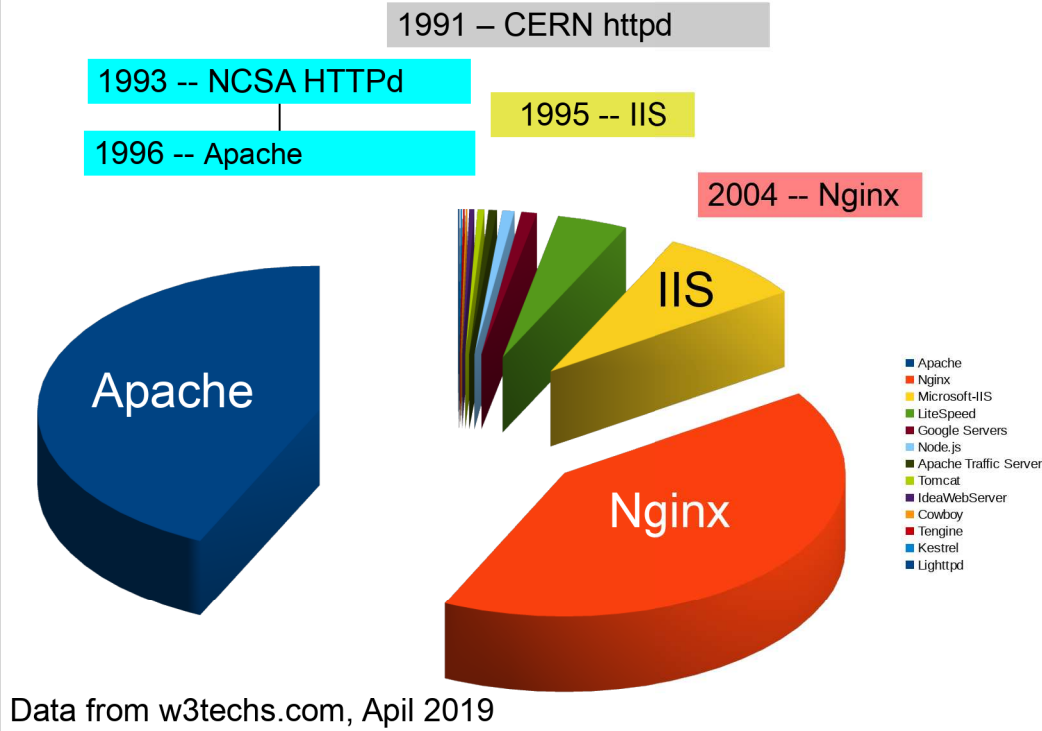


This is the world's first web server, a NeXT computer owned by Tim Berners-Lee, while he was at CERN.

NeXT was a company founded by Steve Jobs, after he was temporarily kicked out of Apple in the 1980s. Jobs referred to this machine as “3M”: it had a Megabyte of memory, a Megapixel display and a Megaflop of computing power. The design of the NeXT operating system was very influential on the Macintosh after Jobs returned to Apple.

The first web browser, also written by Tim Berners-Lee, was written for the NeXT operating system, and could only run on these computers.

Web Server History and Market Share:



Apache, IIS, and Nginx dominate the market. Apache is the most widely used, so we'll be talking exclusively about the Apache web server.

Apache is also available for Windows, and is included as part of Mac OS X.

The latest contender is “nginx”, a fast, lightweight web server originally developed for Russian search engine “Rambler”.

“httpd”, and Apache Configuration Files:

Most of the configuration files for Apache usually live under the `/etc/httpd` directory. The most important of these is `“/etc/httpd/conf/httpd.conf”`.

Here's the beginning of a typical `httpd.conf`:

<code>ServerRoot "/etc/httpd"</code>	←	Default place for log files and additional configuration files.
<code>StartServers 8</code>	}	Apache starts up several copies of itself to serve incoming requests. These directives control the creation and lifetime of these child processes.
<code>MinSpareServers 5</code>		
<code>MaxSpareServers 20</code>		
<code>ServerLimit 256</code>		
<code>MaxClients 256</code>		
<code>MaxRequestsPerChild 4000</code>		
<code>Listen 80</code>	←	The port on which the server will listen.
<code>User apache</code>	}	All Apache processes will be owned by this user and group. E.g., for display in error messages.
<code>Group apache</code>		
<code>ServerAdmin root@localhost</code>	←	
<code>DocumentRoot "/var/www/html"</code>	←	The directory under which our web documents live.
<code>DirectoryIndex index.html home.html</code>	←	A list of default documents to look for, if none is specified in the URL.

The Apache server is started as a service called “httpd”:

```
[root@demo ~]# service httpd start
```

```
[root@demo ~]# httpd -t ← You can use this command to check the syntax of your Apache config files.
```

The `httpd.conf` file is a text file containing single-line “directives” that control Apache's behavior.

All files served up by this web server must be readable by whichever user you specify with the “User” directive.

The “`httpd -t`” command won't do anything except check the syntax of the configuration files.

The main apache server program is usually `/usr/sbin/httpd`.

For full information about Apache configuration directives, see:

<https://httpd.apache.org/docs/2.4/configuring.html>

Scope of Directives, and User Directories:

Here's another excerpt from httpd.conf, showing how the scope of a set of directives can be limited to a particular directory:

```
<Directory "/var/www/html">
  Options FollowSymLinks
  AllowOverride None
  Order allow,deny
  Allow from all
</Directory>

UserDir public_html

<Directory /home/*/public_html>
  Options FollowSymLinks Includes
  AllowOverride AuthConfig Indexes FileInfo
</Directory>
```

The diagram shows two code blocks from httpd.conf. The first block is enclosed in `<Directory "/var/www/html">` and contains `Options FollowSymLinks`, `AllowOverride None`, `Order allow,deny`, and `Allow from all`. A blue callout box with an arrow points to this block, containing the text: "These directives only apply to files under "/var/www/html"". The second block is `<Directory /home/*/public_html>` and contains `Options FollowSymLinks Includes`, `AllowOverride AuthConfig Indexes FileInfo`. A second blue callout box with an arrow points to this block, containing the text: "These apply to files in /home/*/public_html".

The “**Options**” directive sets various options for this directory.

The “**AllowOverride**” directive controls which system-wide options users are allowed to change in their own configuration files.

The “**UserDir**” directive sets the name of a personal web directory underneath each user's home directory. This can be disabled by saying “UserDir disabled”.

By default, directives apply to all documents on the server, but the scope of a directive can be limited by placing it between delimiters that specify a particular directory, URL, filename pattern, etc.

We'll see an example later where some directives only apply to files with names matching a particular pattern.

Controlling Access:

The “Allow”, “Deny” and “Order” directives can be used to control who has access to your web server, based on IP address or host name.

This would allow hosts in the domain “example.com”, but reject all others:

```
Order allow,deny
Allow from example.com
```

More examples:

```
Allow from example.com
Allow from mydomain.net mydomain.org
Allow from 192.168.5.3
Allow from 192.168.0.0/255.255.0.0
Allow from mypc.example.com
Deny from badpeople.com
```

Order Allow,Deny

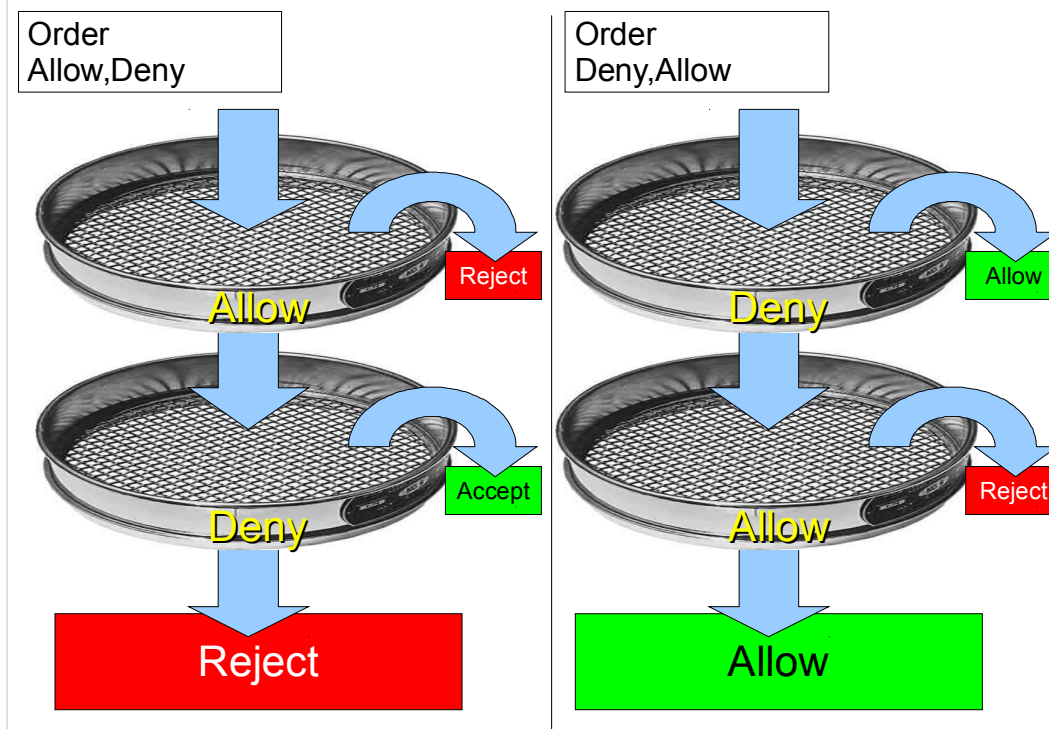
First, all Allow directives are evaluated; at least one must match, or the request is rejected. Next, all Deny directives are evaluated. If any matches, the request is rejected. Last, any requests which do not match an Allow or a Deny directive are denied by default.

Order Deny,Allow

First, all Deny directives are evaluated; if any match, the request is denied unless it also matches an Allow directive. Any requests which do not match any Allow or Deny directives are permitted.

The descriptions of “Order Allow, Deny” and “Deny,Allow”, at the bottom of this slide, are taken directly from the Apache documentation. Maybe it's just me, but I've always found they make my head hurt. That's why I came up with the diagram in the next slide.

How the “Order” Directive Works:



Here's a graphical representation of the description on the preceding page. You can think of “Allow” and “Deny” as two sieves. If an address matches an Allow rule, for example, it falls through that sieve. Otherwise it's caught. The address of an incoming connection is filtered through these sieves until it's allowed or rejected.

One thing to notice: you can determine the default behavior by looking at the last item in the “Order” statement. When you say “Order Allow,Deny”, the effect is to deny incoming connections unless they're explicitly allowed. When you say “Order Deny, Allow”, the effect is to accept incoming connections, unless they're explicitly denied.

Determining the file types of documents:

When Apache sends a document to a web client, it passes along a file type describing the file. These types are taken from the list of "Multimedia Internet Mail Extensions" (MIME) types. The Apache **"TypesConfig"** directive points at a local file that maps MIME types to file extensions.

```
TypesConfig /etc/mime.types
DefaultType text/plain
MIMEMagicFile conf/magic
```

DefaultType is used if the file's type can't be determined.

application/msword	doc
application/pdf	pdf
application/postscript	ai eps ps
application/vnd.ms-excel	xls
audio/mpeg	mpga mp2 mp3
image/gif	gif
image/jpeg	jpeg jpg jpe
text/html	html htm
text/plain	asc txt
etc...	

The mime.types file consists of a list of MIME-types and associated file extensions.

If the file's type can't be determined by extension, apache can try the "magic" file. This is a list of byte-combinations to look for in a file, with matching MIME types.

```
0 string \376\067\0\043 application/msword
0 string \333\245-\0\0\0 application/msword
0 string %! application/postscript
0 string \004%! application/postscript
0 string %PDF- application/pdf
etc...
```

The “Include” Directive and .htaccess Files:

The `httpd.conf` file can refer to other configuration files through the “**Include**” directive. For example, the following line would include all the files matching `/etc/httpd/conf.d/*.conf`:

```
Include conf.d/*.conf
```

The directives in each of these files would be parsed just as though they were typed into the main `httpd.conf` at the point where the Include directive appears.

Also, directories can contain local configuration files that supplement or override the directives in the main Apache configuration files. These files are usually named “.htaccess”, but their name can be set using the `AccessFileName` directive in one of the main configuration files:

```
AccessFileName .htaccess
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

.htaccess files are often used to password-protect web documents. It's a security risk to allow the .htaccess file itself to be visible over the web. Here we use the “Files” scope to restrict access to any file with a name beginning in “.ht”.

This shows how other files, besides `httpd.conf`, can influence the behavior of Apache.

Password-Protecting Pages:

A user can password-protect documents in a web directory by creating a .htaccess file in that directory containing Apache directives like the following:

```
AuthUserFile /home/elvis/htpasswd
AuthName "Sekrit Stuff"
AuthType Basic

require valid-user
```

This will cause Apache to require visitors to supply a valid username and password, matching an entry in the user's "AuthUserFile".

The user can create the AuthUserFile and add a user to it using commands like this:

```
~/demo> touch /home/elvis/htpasswd
~/demo> htpasswd /home/elvis/htpasswd webuser
New password:
Re-type new password:
Adding password for user webuser
```

For this to work, the directory will need to be marked with "AllowOverride AuthConfig" in the system-wide Apache configuration.

The file /home/elvis/htpasswd will look like this:

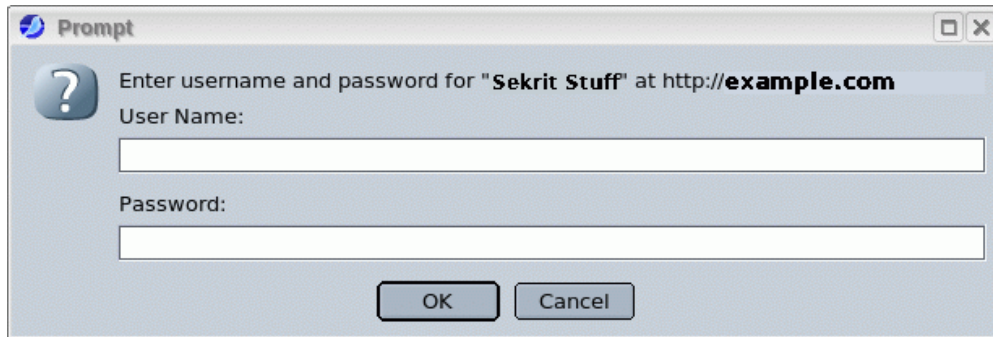
```
webuser:F3t8MGoKLLuH.
```

Note that, as in the example above, the user can have his or her own "htpasswd" file. It can actually have any name, as long as it matches the name given to "AuthUserFile". The htpasswd file should always live in a directory that's NOT available through the web. For example, if a user has web files under "public_html", the htpasswd file could be stored above that, directly in the user's home directory.

In the example above, we've created an entry for the user name "webuser", but we could use any name. You can add more username/password pairs to the htpasswd file by repeatedly using the htpasswd command. You can use the "require" directive to require either "valid-user" (any user) or a specific user (e.g., "require user elvis").

Username and Password Prompt from Browser:

After a directory is password-protected, browsers will require that you enter a valid username and password (as defined in the associated .htaccess file) to get access to the directory.



Apache Modules:

Extra features can be added to Apache through plug-in modules. Most Linux distributions will have many popular Apache modules pre-installed. To use a module, it must be loaded through a “LoadModule” directive in httpd.conf or another Apache configuration file.

```
LoadModule auth_basic_module modules/mod_auth_basic.so
LoadModule include_module modules/mod_include.so
LoadModule cgi_module modules/mod_cgi.so
```

An index of official Apache modules can be found here:
<http://httpd.apache.org/docs/2.4/mod/>

This shows three important Apache modules:

auth_basic handles the basic authentication we saw earlier, mod_include enables Server-Side Includes, which we'll talk about in the next section, and mod_cgi enables server-side scripts. Unfortunately, we don't have time to talk about the last of these today.

Part 4: Server-Side Includes (SSI)



Server-Side Includes (SSI) are a very useful mechanism that can make web pages more flexible and easier to maintain. SSI gives you the ability to use additional markup tags in your HTML files. The SSI tags are evaluated by the web server, and are never passed to a visitor's browser (if SSI is working properly). Hence the term "Server-Side".

What Does SSI Do?

SSI adds the following abilities to otherwise static HTML documents:

- **Lets you dynamically include the content of other files.**

You can, for example, have a standard header and footer that you put at the top and bottom of each of your pages using the SSI include statement.

- **Gives you the ability to do “if” statements.**

You can decide whether or not to display a section of your HTML based on the values of several pre-defined variables, or variables defined by you.

- **Lets you display the values of some pre-defined variables.**

Some of these hold the current time and date and the file's last modification time.

- **Lets you define and use your own variables.**

These can be defined in the current document, another included document, or in your .htaccess file.

For more information about SSI, see: <http://httpd.apache.org/docs/2.4/howto/ssi.html>

Enabling Server-Side Includes:

To enable SSI, you'll need to do the following:

- Make sure the `mod_include` module is loaded in your Apache configuration.
- The “`Options Includes`” directive must be given in `httpd.conf`, or the directive “`Options +Includes`” needs to be in the directory's `.htaccess` file.
- You'll need to tell Apache **which files** should have server-side includes enabled. You can do this with a directive like:

```
AddOutputFilter INCLUDES .html
```

This tells Apache that any file with a name ending in “.html” should be processed through the SSI module. This will take some extra processing time for each file, so if performance is a concern you may want to use a different file extension (like “.shtml”) for the files that will use SSI.

There's also a mechanism called “XBitHack”, which will cause Apache to enable SSI only for HTML files whose “execute” permission bit has been set. For more information about this, see:

<http://httpd.apache.org/docs/2.4/howto/ssi.html>

Including a File with SSI:

The syntax for SSI "include" statements is shown below. The included files **can include others**, and so on. The contents of the included files will be inserted into the HTML dynamically, before the web server delivers the document to a visitor.

```
<html>
<head>
  <title>This is a web page!</title>
</head>
<body>
  <!--#include virtual="header.html" -->
  This is some text. It will appear in the web page.
  Here is <a href="http://example.com">a link</a> to
  another page.

  Here's an image:
  

  The end.
  <!--#include virtual="footer.html" -->
</body>
</html>
```

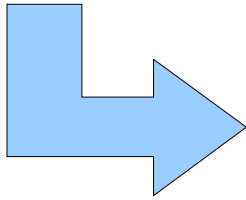
This space is mandatory.

SSI Variables:

The following simple HTML file uses the SSI “printenv” command to generate a list of all current SSI variables and their values:

ssitest.html

```
<html>
<body>
<pre>
  <!--#printenv -->
</pre>
</body>
</html>
```



The variables shown here were all automatically generated by Apache.

Output:

```
SERVER_SOFTWARE=Apache/2.2.3 (CentOS)
SERVER_NAME=example.com
SERVER_ADDR=192.168.100.106
SERVER_PORT=80
REMOTE_ADDR=192.168.4.8
DOCUMENT_ROOT=/home/httpd/html
SERVER_ADMIN=root@example.com
SCRIPT_FILENAME=/home/httpd/html/compfac/junk.html
REMOTE_PORT=56304
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/products/ssitest.html
SCRIPT_NAME=/products/ssitest.html
DATE_LOCAL=Wednesday, 01-Apr-2009 20:32:22 EDT
DATE_GMT=Thursday, 02-Apr-2009 00:32:22 GMT
LAST_MODIFIED=Wednesday, 01-Apr-2009 20:31:51 EDT
DOCUMENT_URI=/products/ssitest.html
USER_NAME=root
DOCUMENT_NAME=ssitest.html
etc...
```


Conditional Statements:

The following example shows how to use an SSI “if” statement to test the value of the “**REMOTE_ADDR**” variable, which contains the IP address of the client computer that’s connecting to the web server:

```
<html>
<body>
<pre>
  <!--#if expr="$REMOTE_ADDR = /^192\.2\.\./" -->
  Hi there!
    This text is only visible to browsers
    on the 192.2.*.* network.
  <!--#endif -->
</pre>
</body>
</html>
```

In this example, we test the value against a regular expression. Apache SSI regular expressions have the same syntax as those in the Perl scripting language.

We could use this, for example, to display different versions of a web page to internal or external visitors. We might make a departmental web page that has additional information that's only visible to people within the department.

Setting and Echoing SSI Variables:

We can set the value of a variable, and then use that value later:

test.html

```
<html>
<body>
<!--#set var="bestcolor" value="fuchsia" -->
<pre>
  The best color is <!--#echo var="bestcolor" -->
</pre>
<font color="<!--#echo var="bestcolor" -->">
Here is some text in this beautiful color.
</font>
</body>
</html>
```

Alternatively, we could set the value in the directory's .htaccess file, using the Apache **SetEnv** directive:

```
SetEnv bestcolor "fuchsia"
```

Note that if you set the variable in a directory's .htaccess file, then the variable will be available to all pages underneath that directory, automatically.

Also note that it's sometimes convenient to define variables in other files, that are then "include"ed via SSI's `<!--#include` command.

Part 5: Cookies



HTTP connections are “stateless”, meaning that the web server doesn't remember who you are from one request to the next. It's like a person with severe short-term memory loss, who needs to be constantly reminded who you are.

HTTP cookies provide web servers with a mechanism for storing information about visitors. Instead of storing the information in the web server itself, though, the information is sent to the visitor's browser, and the browser caches the information on the visitor's computer.

HTTP Headers:

When you get a document from a web server, the server sends header information along with the document. Normally, these header lines aren't seen by the user. They're used behind the scenes by the browser, invisibly.

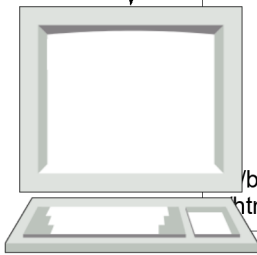


```
Date: Thu, 02 Apr 2009 02:08:03 GMT
Server: Apache/2.2.3 (CentOS)
Accept-Ranges: bytes
Connection: close
Content-Type: text/html
```

```
<html>
<head>
  <title>This is a web page!</title>
</head>
<body>
  This is some text. It will appear in the web page.
  Here is <a href="http://example.com">a link</a> to
  another page.

  Here's an image:
  

  The end.
</body>
</html>
```



As we saw last time, you can see these headers if you telnet to port 80 on the web server and manually use the HTTP “GET” command. You can also see them if you use the “-S” flag when you fetch a document with wget.

Setting Cookies:

The HTTP header information can contain "cookies". A cookie is just a piece of data that the web server gives to the browser.

```
Date: Thu, 02 Apr 2009 02:08:03 GMT
Server: Apache/2.2.3 (CentOS)
Accept-Ranges: bytes
Connection: close
Content-Type: text/html
Set-Cookie: VisitorNumber=123456
Set-Cookie: FavoriteColor=blue; Expires=Wed, 09 Jun
2021 10:18:14 GMT
```

```
<html>
<head>
  <title>This is a web page!</title>
</head>
<body>
  This is some text. It will appear in the web page.
  Here is <a href="http://example.com">a link</a> to
  another page.

  Here's an image:
  

  The end.
</body>
</html>
```

When the browser receives a cookie, the cookie is **stored on the user's computer**.

Cookies are grouped according to the server that sent them.

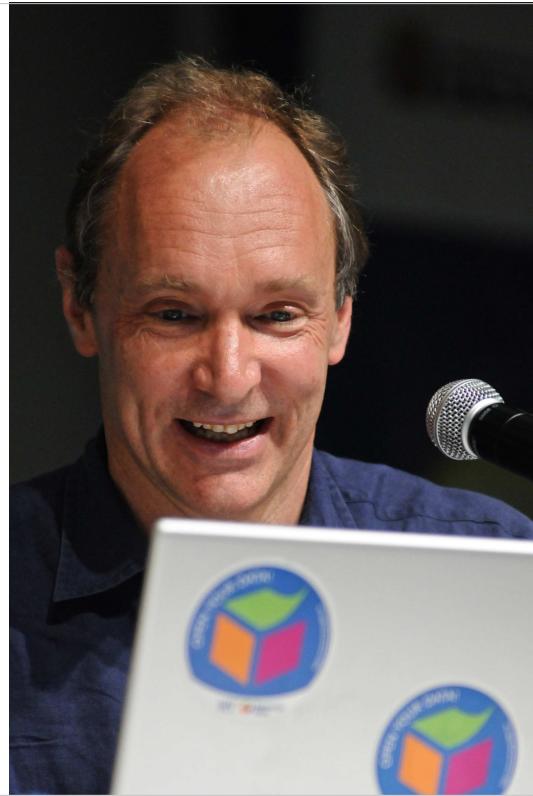
Whenever the browser contacts the server again, it tells the server the values of any cookies associated with that server.

Cookies give a web server a way to store information about a user. The data may include user preferences, or a unique identifier for the user. Instead of storing these values on the server, they're stored on the client side, in each user's browser.

Cookies can optionally include an expiration date. In the absence of a specified expiration date, the cookie will be deleted by the browser at the end of the current session (i.e., when you exit from the browser). Expiration dates are specified like this:

```
Set-Cookie: name=value; Expires=Wed, 09 Jun 2021
10:18:14 GMT
```

Part 6: History



Tim Berners-Lee at Campus Party Brasil, 2009, photo by Silvio Tanaka.

Tim Berners-Lee invented the World Wide Web in 1991, while working at CERN. Today, many people think of the Web and the Internet as almost synonymous. But the internet began with the first ARPANET links in 1969. What was happening in the 22 years between the birth of the internet and the birth of the Web?

The Yanoff List:

In the early 90s, Scott Yanoff maintained a list of “internet services”. Here's an excerpt from a 1991 list, showing the range of resources that existed before the web:

```
-CARL                telnet pac.carl.org or 192.54.81.128
  offers: Online database, book reviews, magazine fax delivery service.

-Cleveland Freenet   telnet freenet-in-a.cwru.edu or 129.22.8.82
  offers: USA Today Headline News, Sports, etc...

-Ham Radio Callbook  telnet marvin.cs.buffalo.edu 2000 or 128.205.32.4
  offers: National ham radio call-sign callbook.

-IRC Telnet Client   telnet bradenville.andrew.cmu.edu or 128.2.54.2
  offers: Internet Relay Chat access, like a CB on the computer.

-Library of Congress telnet dra.com or 192.65.218.43
  offers: COPY of Library of Congress (Assumes terminal is emulating a vt100).

-Lyric Server        ftp vacs.uwp.edu
  offers: Lyrics in text file format for anonymous ftp downloading.

-NASA SpaceLink      telnet spacelink.msfc.nasa.gov or 128.158.13.250
  offers: Latest NASA news, including shuttle launches and satellite updates.

-PENpages            telnet psupen.psu.edu or telnet 128.118.36.5
  offers: Agricultural info (livestock reports, etc.) (Login: PNOTPA)

-Weather Service     telnet madlab.sprl.umich.edu 3000 or 141.212.196.79
  offers: Forecast for any city, current weather for any state, etc.
```

By the 1980s the Internet was a bustling community, with many useful resources. These were available through Usenet newsgroups, mailing lists, anonymous FTP servers, or telnet.

Anonymous FTP Archives:

Here's a small part of a list of anonymous-ftp file archives posted by Edward Vielmetti to the comp.archives newsgroup in 1989. Comp.archives was a newsgroup devoted to announcements and discussion of software archives.

```
a.cs.uiuc.edu # parallel gnumake, TeX, dvi2ps, gif
accuvax.nwu.edu # PibTerm 4.1.3
ahwahnee.stanford.edu # pcip interface specs
ai.toronto.edu # Canada, SunOS4.0 SLIP beta, lots of stuff
albanycs.albany.edu # best of comp.graphics
allspice.lcs.mit.edu # RFC1056 (PCMAIL) stuff
ames.arc.nasa.gov # pcrn, gnu grep
games
arisia.xerox.com # lisp, tcp/ip, IDA sendmail kit
arizona.edu # Icon, SR, SBProlog languages
arthur.cs.purdue.edu # RCS, Purdue tech reports
athena-dist.mit.edu # Hesiod name server, Kerberos
brownvt.brown.edu # MAC
bu-it.bu.edu # bunches of stuff
bugs.nosc.mil # Minix
c.isi.edu # info-ibmpc (Tenex)
cadre.dsl.pittsburgh.edu # jove for the Mac
camelot.berkeley.edu # "pmake", yet another parallel make
cayuga.cs.rochester.edu # LaTeX styles, Jove, NL-KR mailing list
celray.cs.yale.edu # ispell, dictionary
(tenex)
charon.mit.edu # perl+patches, xdvi
cheops.cis.ohio-state.edu # comp.sources.*, alt.sources
citi.umich.edu # pathalias, CITI MacIP, webster
clutx.clarkson.edu # Turbo C stuff, net kit
cs.cmu.edu # screen
cs.orst.edu # Xlisp
cs.utah.edu # A Tour of the Worm
cunix.cc.columbia.edu # MM mail user agent beta, Kermit
cygnusx1.cs.utk.edu # GCC, MM, Scheme
decwrl.dec.com # no FTP; gatekeeper.dec.com
games
devvax.tn.cornell.edu # tn3270, gated
drizzle.cs.uoregon.edu # raytracing archive (markv)
expo.lcs.mit.edu # a home of X, portable bitmaps
f.ms.uky.edu # lots of interesting things
flash.hellcore.com # Karn's RFC and IEN collection
gatekeeper.dec.com # X11, recipes, cron, map, Larry Wall stuff

trwind.ind.trw.com # Turbo C src for net.exe
tutum.cs.umd.edu # News pd software
tut.cis.ohio-state.edu # GNU, lots of interesting things
ucbarpa.berkeley.edu # tn3270, pub/4.3
ucbvax.berkeley.edu # nntp, gnews, awm
ucsd.edu # KA9Q archives, packet driver
umn-cs.cs.umn.edu # Sendmail and related, vectrex

unmvax.unm.edu # getmaps,
utadnx.cc.utexas.edu # VMS sources (zetaps, laser, sxlps
uunet.uu.net # usenet archives
uxc.cso.uiuc.edu # games, misc, HitchHiker's Guide
uxe.cso.uiuc.edu # amiga/Fish disks, PC-SIG 1-499
vax.ftp.com # FTP software, inc.
venera.isi.edu # statspy (NNstat)
venus.ycc.yale.edu # SBTEx
vgr.br1.mil # bsd ping + record route
venera.isi.edu # GNU Chess
watmath.waterloo.edu # lots of stuff
wsmr-simtel20.army.mil # MS-DOS, Unix, CP/M, Mac, lots!

zap.mit.edu # original X11 distribution
zaphod.ncsa.uiuc.edu # NCSA Telnet source, binaries

trwind.ind.trw.com # Turbo C src for net.exe
tutum.cs.umd.edu # News pd software
tut.cis.ohio-state.edu # GNU, lots of interesting things
ucbarpa.berkeley.edu # tn3270, pub/4.3
ucbvax.berkeley.edu # nntp, gnews, awm
ucsd.edu # KA9Q archives, packet driver
umn-cs.cs.umn.edu # Sendmail and related, vectrex

unmvax.unm.edu # getmaps,
utadnx.cc.utexas.edu # VMS sources (zetaps, laser, sxlps
uunet.uu.net # usenet archives
uxc.cso.uiuc.edu # games, misc, HitchHiker's Guide
uxe.cso.uiuc.edu # amiga/Fish disks, PC-SIG 1-499
```

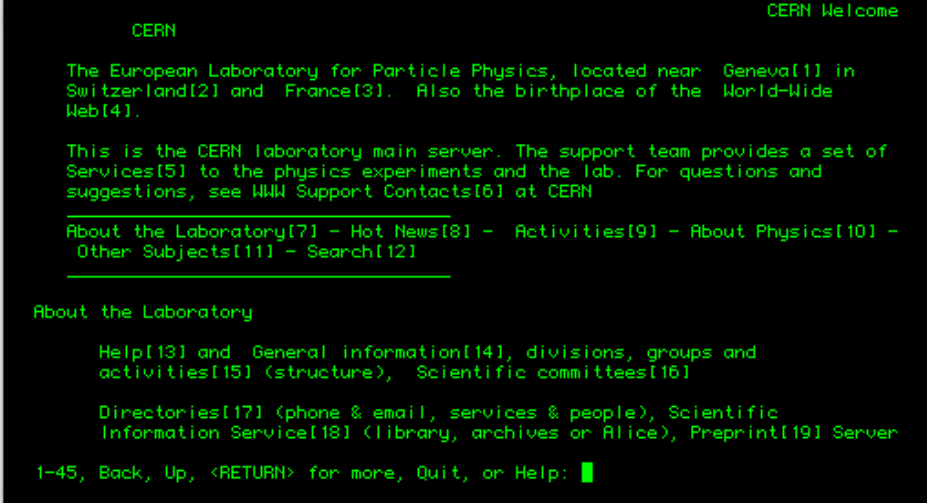
Even though there were many resources that the average non-technical person would find useful, or at least interesting, on the Internet, there were technical, practical and social barriers that made it hard for the average person to use these resources.

Microsoft Windows computers at that time didn't even support the TCP/IP protocols natively. In order to have any hope of using the Internet, Windows users would need to find and install third-party software.

Then they'd need to learn (and remember) how to use FTP, telnet and other programs, and somehow locate (and remember) the locations and other necessary information to connect to the resources they were interested in.

Telnet:

This is what CERN's web site looked like to many people in the early 1990s. Anyone in the world could type "telnet info.cern.ch" and they'd be presented with a text-based interface to this thing called the "World Wide Web".



```
CERN                                     CERN Welcome

The European Laboratory for Particle Physics, located near Geneva[1] in
Switzerland[2] and France[3]. Also the birthplace of the World-Wide
Web[4].

This is the CERN laboratory main server. The support team provides a set of
Services[5] to the physics experiments and the lab. For questions and
suggestions, see WWW Support Contacts[6] at CERN

-----
About the Laboratory[7] - Hot News[8] - Activities[9] - About Physics[10] -
Other Subjects[11] - Search[12]
-----

About the Laboratory

Help[13] and General information[14], divisions, groups and
activities[15] (structure), Scientific committees[16]

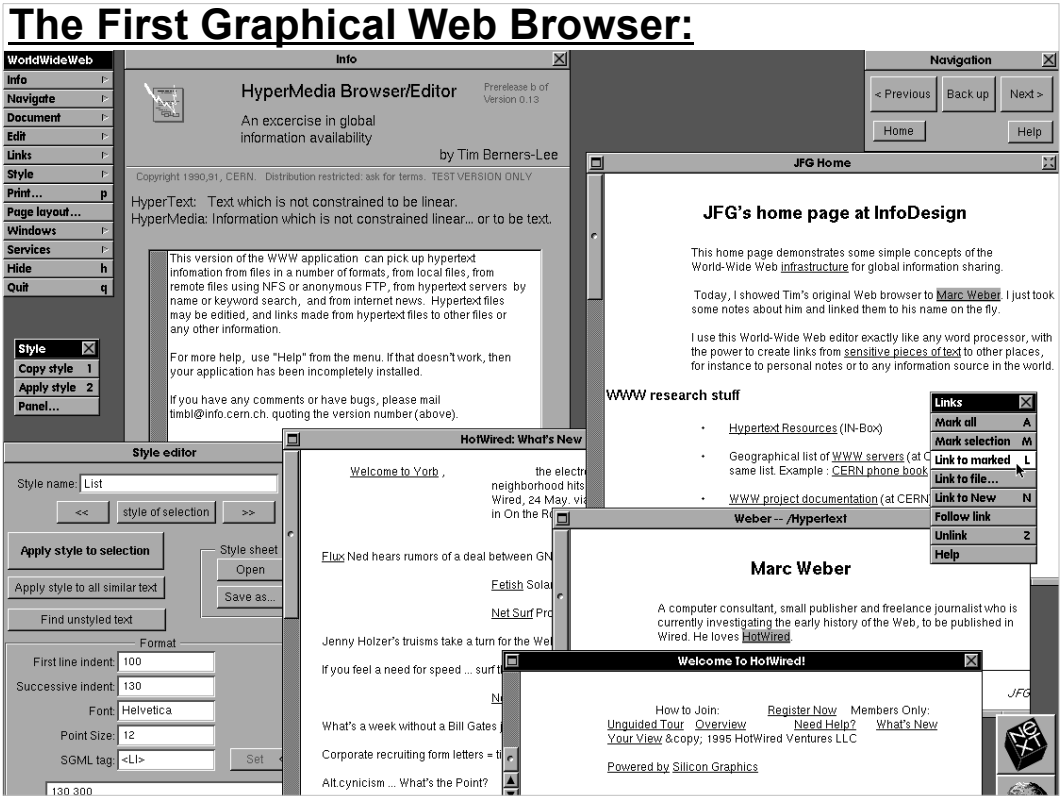
Directories[17] (phone & email, services & people), Scientific
Information Service[18] (library, archives or Alice), Preprint[19] Server

1-45, Back, Up, <RETURN> for more, Quit, or Help: █
```

This is the first view I had of the World Wide Web. There were lots of similar things popping up at the time: WAIS,archie, gopher, etc. The WWW was another attempt at making Internet resources easy to find. I looked at it and thought, "Well, this'll never amount to anything."

Then, in the 1990s, along came Web browsers. In particular, NCSA Mosaic suddenly made it easy for Windows users to access the treasure-trove of resources on the Internet. Mosaic supported the new-fangled World Wide Web, but it also supported the older protocols like FTP. A world of resources was just a click away.

The new HTML file format made it easier, going forward, to create new resources that were easy for non-technical people to use, but that just greased the skids. The web browser was the engine that powered the Web revolution.



This is Tim Berners-Lee's original graphical web browser (and editor) for the NeXT. Berners-Lee invented the web server, the HTTP protocol, and the HTML file format. None of these would have made much difference to the average person without the web browser, though.

Even without the other things, I think we'd still have ended up more or less where we are today, given some sort of browser capable of connecting unskilled users with the resources that were already on the internet in the 1980s.



Thanks!