Installation

Installing NCSA httpd can be broken down into these basic steps.

Downloading NCSA httpd

If you already have a copy of the current release of NCSA httpd (1.3), you may skip to the next step.

To download NCSA httpd, you must first decide on a binary.

NCSA provides binaries for the types of systems we have available to us. Check this list to see if your system is one of these.

Precompiled Binaries

Select the item which matches your system to download a copy of the software.

You will get a .tar.Z file with the server binary, and other support directories. From Mosaic you can simply download the file. Here are the URLs

- file://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current/httpd_sgi.tar.Z

  for Silicon Graphics, Inc.
  System: IRIS Crimson VGXT, IRIX 4.0.5C.

- file://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current/httpd_sun4.tar.Z

  for Sun Microsystems
  System: SPARCserver 690MP, SunOS 4.1.3.

- file://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current/httpd_decmips.tar.Z

  for Digital Equipment Corporation
  DECstation 5000, Ultrix 4.2 Rev. 96.

- file://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current/httpd_decaxp.tar.Z

  for Digital Equipment Corporation (AXP)
  DEC 3000 AXP Model 500, OSF/1 1.3.

- file://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current/httpd_rs6000.tar.Z

  for International Business Machines
  IBM RS/6000 Model 550, AIX 3.2.4.

- file://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current/httpd_hp.tar.Z

  for Hewlett−Packard

HP 9000 model 730, HP−UX 9.01.

If your system is not on this list, or if you feel more comfortable doing so, you must compile a binary.

Compiling NCSA httpd

First, download the source from

file://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current/httpd_source.tar.Z

and use `tar` and `uncompress` to decode it. For example, make a directory and move the httpd_source in it. Then type the following commands.

% uncompress filename.tar.Z
% tar −xvf filename.tar

In compiling NCSA httpd, you must compile three things: the server, the scripts, and the support programs.

The Makefiles for these things should be easy to edit. The important step is getting the AUX_CFLAGS in the server Makefile (src/Makefile) right. If your system has an AUX_CFLAGS associated with it, use it.

If not, you must edit `src/httpd.h`. It is *important* that you get the setting of the BSD flag correct.

Later on in the installation, when you run httpd, you may see Night of the Living Dead on your machine, as defunct processes pile up and eventually bring it to a grinding halt. If you see this, you have the wrong setting of BSD.

Configuration

Configuration means customizing httpd's configuration files to reflect your system and how httpd should act on your system.

Please read this general information which applies to all of httpd's configuration files.

- Case insensitive

  Except where pathnames are involved, these files are not case sensitive.

- Comment lines begin with #

  Lines which should be ignored begin with #, the hash sign. This must be the first character on the line. Comments must be on a line by themselves.

- One directive per line

  Each line of these files consists of:

```
Directive data [data2 ... datan]
```

`Directive` is a keyword httpd recognizes, followed by whitespace. `data` is specific to the directive. Any additional data entries should be separated by whitespace.

- Extra whitespace is ignored

  You can put extra spaces or tabs between `Directive` and `data`. To embed a space in `data` without separating it from any subsequent arguments use a \ character before the space.

There are three main configuration files you must edit.

Server Configuration File

When you unpacked httpd, there should have been a directory called `conf`. In there is a file called `httpd.conf-dist`. This is a template for your server configuration file. We suggest you use this file only for reference, and edit the file `conf/httpd.conf` as your server configuration file.

Here is a list of the directives this file recognizes. Following the link will give you usage information, default values, and examples.

- ## ServerType directive

  Purpose

  The ServerType directive sets how the server is executed by the system.

  Syntax

  `ServerType type`

  `type` is one of:

  - `inetd`

    To run the server from the system process `inetd`. The system then does all socket and child management.

  - `standalone`

    To run the server as a daemon process. The server then does all socket and child management.

  Only one ServerType directive is allowed in the configuration file.

  Default

  If you do not specify a ServerType, httpd assumes:

```
ServerType standalone
```

We recommend against running the server from inetd. Here's why

Why should you run the server standalone as opposed to running it from inetd? If your server has a lot of CPU cycles to burn, then there is nothing wrong with running httpd from inetd. Most servers don't.

If you run the server from inetd, that means that whenever a request comes in, inetd must first fork a new process, then load the httpd binary. Once the httpd binary is loaded, httpd must then load and parse all of its configuration files (including httpd.conf, access.conf, and mime.types), which is quite a task to be done for each and every request that comes in.

Now, contrast this with running standalone. When httpd gets a request, it makes a copy of itself (which requires no loading of a binary since shared text pages are used), and the copy handles the request. The configuration files have already been loaded on startup, and so we don't reload them every time.

Examples

```
ServerType standalone
```

```
ServerType inetd
```

- ## Port directive

Purpose

The Port directive sets what port httpd listens to for clients.

Syntax

```
Port num
```

`num` is a number from 0 – 65536. Most ports below 1024, except port 80, are reserved by the system.

If you want to use a port below 1024 (such as 80, the standard HTTP port), and your ServerType is `standalone`, you will need to run httpd as root when starting it up.

Only one Port directive is allowed in the configuration file.

Default

If you do not specify a Port, httpd assumes:

```
Port 80
```

Examples

`Port 8080`

The server would listen to port 8080.

`Port 84`

The server would try to listen to port 84.

- ## User directive

This directive is only applicable if you are using a ServerType of `standalone`.

Purpose

The User directive sets which user id the server will answer requests as.

In order to use this directive, the standalone server must be initially run as `root`.

This directive does not mean that the original daemon process will run as the given user, it means that the children which answer requests run as the given user.

Syntax

`User id`

`id` is one of:

- A name

  Refer to the given user by name.

- # followed by a user number

  Refer to a user by their number.

Only one User directive is allowed in the configuration file.

Default

If you do not specify a User, httpd assumes:

`User #-1`

Examples

`User nobody`

This will cause the server to run as user nobody.

`User #-2`

This will cause the server to run as user number −2.

- ## Group directive

This directive is only applicable if you are using a ServerType of `standalone`.

Purpose

The Group directive sets which group id the server will answer requests as.

In order to use this directive, the standalone server must be initially run as `root`.

This directive does not mean that the original daemon process will run as the given group, it means that the children which answer requests run as the given group.

Syntax

`Group id`

`id` is one of:

- A name

  Refer to the given group by name.

- # followed by a group number

  Refer to a group by its number.

  This `id` must be a valid group found in `/etc/group`, or else a runtime server error will occur.

Only one Group directive is allowed in the configuration file.

Default

If you do not specify a Group, httpd assumes:

`Group #-1`

Examples

`Group nogroup`

This will cause the server to run as group `nogroup`.

`Group #65536`

This will cause the server to run as group number 65536.

- ## ServerAdmin directive

Purpose

The ServerAdmin directive gives the server your e−mail address. This is so that the server can give people your address to report error conditions.

Syntax

```
ServerAdmin address
```

Where `address` is an e−mail address.

Only one ServerAdmin directive is allowed in the server configuration file.

Default

If you do not specify a ServerAdmin, httpd assumes nothing and prints no address for reporting errors.

Examples

```
ServerAdmin www@widget.com
```

Errors will have the address www@widget.com as the person to blame.

- ## ServerRoot directive

Purpose

The ServerRoot directive sets the directory in which httpd lives.

Upon startup, httpd expects to find the Server Configuration File as conf/httpd.conf in the ServerRoot directory.

Other Server Configuration directives may use this directory to give relative paths for locations of files.

Syntax

```
ServerRoot dir
```

Where `dir` is an absolute path of a directory on your server machine.

Only one ServerRoot directive is allowed in the server configuration file.

Default

If you do not specify a ServerRoot, httpd assumes:

```
ServerRoot /usr/local/etc/httpd
```

Examples

```
ServerRoot /usr/local/httpd
```

This would set ServerRoot to the directory /usr/local/httpd.

```
ServerRoot /web1/http
```

This would set ServerRoot to /web1/http.

- ## ServerName directive

Purpose

The ServerName directive sets the hostname httpd should return when creating redirection URLs.

This directive should be used on systems where `gethostbyname` may not work on the local host, or where the hostname returned should be a DNS alias such as `www.widget.com`.

Syntax

```
ServerName FQDN
```

Where `FQDN` is the full hostname (including domain name) to be returned as the server address.

Only one ServerName directive is allowed in the server configuration file.

Default

If you do not specify a ServerName, httpd retrieves it through system calls.

Example

```
ServerName www.widget.com
```

This would set the server's hostname as www.widget.com, as opposed to monster.widget.com as would be returned from the gethostname system call.

- ## TimeOut directive

Purpose

The TimeOut directive sets the amount of time the server will wait for a client to send its query once connected, or the maximum amount of time the server will spend waiting for a client to accept information.

Syntax

`TimeOut time`

Where `time` is the amount of time in seconds the server should wait.

Only one TimeOut directive is allowed in the configuration file.

Default

If you do not specify a TimeOut, httpd assumes:

`TimeOut 1200`

This is a timeout of twenty minutes.

Example

`TimeOut 600`

This sets the timeout to ten minutes. For large files served to slow network clients, you may need every second of it.

- ## ErrorLog directive

Purpose

The ErrorLog directive sets the file to which httpd will log errors it encounters. It currently logs the following error conditions:

  - Clients which time out
  - Scripts which produce no output
  - .htaccess files which attempt to override things they do not have permission to
  - Server bugs which produce a segmentation violation or bus error
  - User Authentication configuration problems

Syntax

`ErrorLog file`

`file` is the name of the file to which errors will be logged. It is either a full pathname, or a partial pathname relative to ServerRoot.

Only one ErrorLog directive is allowed in the configuration file.

Default

If you do not specify an ErrorLog, httpd assumes:

`ErrorLog logs/error_log`

Examples

`ErrorLog logs/errors`

This logs errors to the file logs/errors in the ServerRoot directory.

`ErrorLog /tmp/httpd-errors`

This logs errors to the file `/tmp/httpd-errors`.

`ErrorLog /dev/null`

This effectively turns off error logging.

- ## TransferLog directive

Purpose

The TransferLog directive tells httpd where to record client accesses.

The logfile format is as follows. Each line consists of:

```
host rfc931 authuser [DD/Mon/YYYY:hh:mm:ss] "request" ddd
bbbb
```

- host: Either the DNS name or the IP number of the remote client
- rfc931: Any information returned by identd for this person, – otherwise.
- authuser: If user sent a userid for authentication, the user name, – otherwise.
- DD: Day
- Mon: Month (calendar name)
- YYYY: Year
- hh: hour (24–hour format, the machine's timezone)
- mm: minutes
- ss: seconds
- request: The first line of the HTTP request as sent by the client.
- ddd: the status code returned by the server, – if not available.
- bbbb: the total number of bytes sent, *not including the HTTP/1.0 header*, – if not available

You can determine the name of the file accessed through `request`.

Syntax

`TransferLog file`

`file` is the name of the file to which transfers will be logged. It is either a full pathname, or a partial pathname relative to ServerRoot.

Only one TransferLog directive is allowed in the configuration file.

Default

If you do not specify a TransferLog, httpd assumes:

```
TransferLog logs/access_log
```

Examples

```
TransferLog logs/downloads
```

This logs transfers to the file logs/downloads in the ServerRoot directory.

```
TransferLog /tmp/httpd-accesses
```

This logs transfers to the file `/tmp/httpd-accesses`.

```
TransferLog /dev/null
```

This effectively turns off transfer logging.

- ## PidFile directive

Purpose

The PidFile directive sets the file to which httpd records the daemon process id.

This directive is used only if ServerType is `standalone`.

Syntax

```
PidFile file
```

`file` is the name of the file to which the process id is logged. It is either a full pathname, or a partial pathname relative to ServerRoot.

Only one PidFile directive is allowed in the configuration file.

Default

If you do not specify a PidFile, httpd assumes:

```
PidFile logs/httpd.pid
```

Examples

```
PidFile logs/pid
```

This logs the process id to the file logs/pid in the ServerRoot directory.

```
TransferLog /tmp/httpd-pid
```

This logs the process id to the file `/tmp/httpd-pid`.

```
PidFile /dev/null
```

The server does not log its process id.

- ## AccessConfig directive

Purpose

The AccessConfig directive gives httpd the location of the global access configuration file.

Syntax

```
AccessConfig file
```

`file` is the name of the global access configuration file. It is either a full pathname, or a partial pathname relative to ServerRoot.

Only one AccessConfig directive is allowed in the configuration file.

Default

If you do not specify an AccessConfig, httpd assumes:

```
AccessConfig conf/access.conf
```

Examples

```
AccessConfig conf/access-global
```

httpd looks for access configuration in the file conf/access−global in the ServerRoot directory.

```
AccessConfig /httpd/admin/access
```

httpd looks for access configuration in the file `/httpd/admin/access`.

- ## ResourceConfig directive

Purpose

The ResourceConfig directive gives httpd the location of the resource configuration file.

Syntax

`ResourceConfig file`

`file` is the name of the resource configuration file. It is either a full pathname, or a partial pathname relative to ServerRoot.

Only one ResourceConfig directive is allowed in the configuration file.

Default

If you do not specify an ResourceConfig, httpd assumes:

`ResourceConfig conf/srm.conf`

Examples

`ResourceConfig conf/resources`

httpd looks for resource configuration in conf/resources in the ServerRoot directory.

`ResourceConfig /httpd/admin/resources`

httpd looks for resource configuration in the file `/httpd/admin/resources`.

- ## TypesConfig directive

Purpose

The TypesConfig directive gives httpd the location of the typing configuration file.

This file is how httpd maps filename extensions to MIME types to return to HTTP/1.0 clients. You should not need to edit it. If you really want to, look at the file format.

Types Configuration File format

> Adventurous types may want to add types to their server for local use. We strongly recommend you use the AddTypes directive (described in section for Resource allocation) instead.

Nonetheless, the format of the types configuration file is as follows.

Each line contains information for one http type. These types resemble MIME types. If you plan to add new ones, you should use subtypes beginning with x−, such as application/x−myprogram.

Lines beginning with # are comment lines, and suitably ignored.

Each line consists of:

`type/subtype ext1 ext2 ... extn`

`type/subtype` is the MIME–like type of the document.

`ext*` is any number of space–separated filename extensions which should be returned to the client if a file with the given extension is referenced.

Syntax

`TypesConfig file`

`file` is the name of the types file. It is either a full pathname, or a partial pathname relative to ServerRoot.

Only one TypesConfig directive is allowed in the configuration file.

Default

If you do not specify an TypesConfig, httpd assumes:

`TypesConfig conf/mime.types`

Examples

`TypesConfig conf/mime-types`

httpd looks for types configuration in the file conf/mime–types in the ServerRoot directory.

`TypesConfig /httpd/admin/types-local`

httpd looks for types configuration in the file `/httpd/admin/types-local`.

- ## IdentityCheck directive

Purpose

This directive enables RFC931 compliant logging of the remote user name for sites which run identd or something similar. This information is logged in access_log. It should not be trusted in any way except for rudimentary usage tracking.

If you do not plan to use this information, keep this directive set to `off`. Your network administrator will thank you later.

Syntax

`IdentityCheck setting`

`setting` is either `on` or `off`.

Only one IdentityCheck directive should appear in the configuration file.

Default

If no IdentityCheck directive is present, httpd assumes:

```
IdentityCheck off
```

Example

```
IdentityCheck on
```

Causes the server to check the identity of each person accessing it.

Resource Configuration

When you unpacked httpd, there should have been a directory called `conf`. In there is a file called `srm.conf-dist`. This is a template for your resource configuration file. We suggest you use this file only for reference, and edit the file `conf/srm.conf` as your resource configuration file.

Here is a list of the directives this file recognizes. This will give you usage information, default values, and examples. Some of the directives refer to wildcard expressions, virtual path or indexing tutorial. Documents for wildcard expressions and the indexing tutorial is provided after the listing of directives. The general document Constructing URL's to your server is provided at the very end and includes more info on the virtual path and setting up your homepage

- ## DocumentRoot directive

  Purpose

  The DocumentRoot directive sets the directory from which httpd will serve files.

  If you need to serve files outside this directory, you can use the Alias directive, or create symbolic links.

  This directive affects how you access files on your server.

  Syntax

  ```
  DocumentRoot dir
  ```

  Where `dir` is an absolute path of the directory you want documents to be served from.

  Only one DocumentRoot directive is allowed in the server configuration file.

  Default

  If you do not specify a DocumentRoot, httpd assumes:

```
DocumentRoot /usr/local/etc/httpd/htdocs
```

Example

```
DocumentRoot /home/web
```

This would set DocumentRoot to the directory /home/web.

- ## UserDir directive

Purpose

The UserDir directive sets the real directory in a user's home directory to index upon recieving a request from a user supported directory.

Syntax

```
UserDir dir
```

Where `dir` is a partial path relative to the users home directory (as given in `/etc/passwd`). The server takes the path it creates with this information, and looks for the user's document in the resulting full path.

The keyword `DISABLED` will disable the user supported directories feature.

Only one UserDir directive is allowed in the server configuration file.

Default

If you do not specify a UserDir, httpd assumes:

```
UserDir public_html
```

Example

```
UserDir web-docs
```

This would set UserDir to web-docs. A request for `/~robm/foo.gif` would cause the server to retrieve `~robm/web-docs/foo.gif`.

- ## DirectoryIndex directive

Purpose

When a client requests a directory, httpd can return a pre-written index, or generate one from the filesystem. The DirectoryIndex directive sets the file httpd should look for as a prewritten index to a given directory.

Syntax

```
DirectoryIndex file
```

Where `file` is a file name.

Only one DirectoryIndex directive is allowed in the server configuration file.

Default

If you do not specify a DirectoryIndex, httpd assumes:

```
DirectoryIndex index.html
```

Example

```
DirectoryIndex .index.html
```

This would set DirectoryIndex to .index.html. A request for `/dir` would cause the server to look for the file `DocumentRoot/dir/.index.html`. If found, the server would send it back to the client. Otherwise, it would create and return an index from the filesystem.

- ## AccessFileName directive

Purpose

When returning a document to a client, the server looks for access control files in the document's directory as well as its parent directories. This directive sets the name of the file httpd should look for to find access control files.

Syntax

```
AccessFileName file
```

Where `file` is a file name.

Only one AccessFileName directive is allowed in the server configuration file.

Default

If you do not specify an AccessFileName, httpd assumes:

```
AccessFileName .htaccess
```

Example

```
AccessFileName .ncsa-acl
```

This would set AccessFileName to .ncsa−acl.

- ## AddType directive

Purpose

Allows you to add entries to the server's default typing information and cause an extension to be a certain type. These directives override any conflicting entries in the TypesConfig file.

Syntax

```
AddType type/subtype extension
```

`type/subtype` is the MIME–like type for the document.

`extension` is the filename extension to map to this type. This can either be a filename extension, a full pathname, or a file name.

You may use as many AddType directives as you wish.

Default

The default types are in the types configuration file.

Example

```
AddType text/plain doc
```

This would cause any file ending in .doc to be served as type text/plain.

- ## AddEncoding directive

Purpose

Allows you to specify an encoding type for a document with a given filename extension.

In order to serve encoded documents to clients, the client must support the given encoding method, as well as the HTTP encoding extension.

Syntax

```
AddEncoding type extension
```

`type` is the encoding type for the document.

`extension` is the filename extension to map to this encoding.

You may use as many AddEncoding directives as you wish.

Default

There are no default encodings.

Example

```
AddEncoding x-gzip gz
```

This would cause any file ending in .gz to be marked as encoded using the x−gzip encoding method.

- ## DefaultType directive

Purpose

If httpd can't type a file through normal means, it will type it as DefaultType.

Syntax

```
DefaultType type/subtype
```

`type/subtype` is the MIME−like type.

Only one DefaultType directive should appear in the configuration file.

Default

If no DefaultType is present, httpd assumes:

```
DefaultType text/html
```

Example

```
DefaultType application/octet-stream
```

This would cause a file with an unknown extension to be returned as type application/octet−stream.

- ## Redirect directive

Purpose

The Redirect directive creates a virtual document on your server, and any accesses to it will be redirected to a new URL.

Syntax

`Redirect virtual URL`

`virtual` is the translated location which should trigger a redirect..

`URL` is the URL of the new document.

Several Redirect directives may appear in the configuration file.

Example

`Redirect /dir1 http://newserver.widget.com/dir1`

This would cause requests for /dir1 to be redirected to the new location, `http://newserver.widget.com/dir1`.

- ## Alias directive

Purpose

The Alias directive creates a virtual document (or directory) on your server. Any accesses to it will be satisfied by a different file or directory.

Syntax

`Alias virtual path`

`virtual` is the translated location of the file/directory.

`path` is the full pathname of the file or directory which should be used to fulfill the request.

Several Alias directives may appear in the configuration file.

Example

`Alias /images /ftp/pub/images`

This would cause requests for `/images` to be satisfied from the directory `/ftp/pub/images`.

Thus, if someone requested `/images/foo.gif`, the server would return `/ftp/pub/images/foo.gif`.

- ## ScriptAlias directive

Purpose

The ScriptAlias directive creates a virtual directory on your server. Any accesses to that virtual directory will be satisfied by returning the output of a CGI server script in that directory.

Syntax

`ScriptAlias virtual path`

`virtual` is the translated location of the script directory.

`path` is the full pathname of the directory which contains server scripts which fulfill the request.

Note: You should always place a trailing / after ScriptAlias directives which reference directories, to prevent similar entries from conflicting with each other.

Several ScriptAlias directives may appear in the configuration file.

Example

```
ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-bin/
```

This would cause requests such as `/cgi-bin/foo` to be satisfied by running the script `/usr/local/etc/httpd/cgi-bin/foo`.

Thus, if someone requested `/cgi-bin/a-script`, the server would run `/usr/local/etc/httpd/cgi-bin/a-script` and send its output to the client.

- ## OldScriptAlias directive

Purpose

The OldScriptAlias directive creates a virtual directory on your server. Any accesses to that virtual directory will be satisfied by returning the output of an NCSA server script in that directory.

Syntax

`OldScriptAlias virtual path`

`virtual` is the translated location of the script directory.

`path` is the full pathname of the directory which contains server scripts which fulfill the request.

Several OldScriptAlias directives may appear in the configuration file.

Example

```
OldScriptAlias /htbin /usr/local/etc/httpd/htbin
```

This would cause requests such as `/htbin/foo` to be satisfied by running the script `/usr/local/etc/httpd/htbin/foo`.

Thus, if someone requested `/htbin/a-script`, the server would run `/usr/local/etc/httpd/htbin/a-script` and send its output to the client.

- ## FancyIndexing directive

Purpose

This directive specifies whether you want fancy directory indexing (with icons and file sizes) or standard directory indexing.

If your documents are being served from an AFS drive, you will want to seriously consider turning fancy indexing off.

Syntax

```
FancyIndexing setting
```

`setting` is either `on` or `off`.

Only one FancyIndexing directive should appear in the configuration file.

Default

If no FancyIndexing directive is present, httpd assumes:

```
FancyIndexing off
```

Example

```
FancyIndexing on
```

Turns fancy indexing on.

- ## DefaultIcon directive

Purpose

This directive specifies what icon should be shown in an automatically generated directory listing for a file which has no icon information.

Syntax

```
DefaultIcon location
```

`location` is the virtual path to the icon on your server.

Only one DefaultIcon directive should appear in the configuration file.

Default

If no DefaultIcon is present, httpd assumes nothing.

Example

```
DefaultIcon /icons/unknown.xbm
```

This would cause a file with no icon to be given the icon /icons/unknown.xbm.

- ## ReadmeName directive

Purpose

This directive specifies what filename httpd should look for when indexing a directory, in order to add a paragraph of description to the end of the index it automatically generates. Generally these paragraphs are used to give a general overview of what's in a directory.

Syntax

```
ReadmeName name
```

`name` is the name of the file httpd should look for when trying to find a description file. httpd will first look for `name.html`, and if found, will display the HTML inlined with its own index. If it finds `name`, it will include the file as plaintext.

Only one ReadmeName directive should appear in the configuration file.

Default

If no ReadmeName is present, httpd assumes nothing.

Example

```
ReadmeName README
```

When generating an index for the directory `/foo`, httpd will look for `/foo/README.html`, and will insert it if found. It will then look for `/foo/README` and insert it if found. If it finds nothing, it will include nothing.

- ## HeaderName directive

Purpose

This directive specifies what filename httpd should look for when indexing a directory, in order to add a custom header. This can describe the contents of the directory.

Syntax

```
HeaderName name
```

`name` is the name of the file httpd should look for when trying to find a description file. httpd will first look for `name.html`, and if found, will display the HTML inlined with its own index. If it finds `name`, it will include the file as plaintext.

Only one HeaderName directive should appear in the configuration file.

Default

If no HeaderName is present, httpd assumes nothing.

Example

```
HeaderName HEADER
```

When generating an index for the directory `/foo`, httpd will look for `/foo/HEADER.html`, and will insert it at the top of the index if found. If not, it will then look for `/foo/HEADER` and insert it if found. If it finds nothing, it will include nothing.

- ## AddDescription directive

Purpose

Tells httpd how to describe a file or a file type while generating a directory index.

Syntax

```
AddDescription "blah blah blah" fileid
```

`fileid` is either a filename extension (like .html), a filename, a full real pathname to a file on your system, or a wildcard pattern to match to filenames.

`blah blah blah` must be surrounded by quotes and is a short (preferably < 1 line) description of the file.

You may use as many AddDescription directives as you wish.

Default

There are no default descriptions.

Example

```
AddDescription "a great batch of nonsense" fargle.bargle
```

A portion of httpd's index for fargle.bargle's directory would look something like this:

 foople.poople (8000 bytes)
 fargle.bargle : a great batch of nonsense (10000 bytes)

- ## AddIcon directive

Purpose

Tells httpd what kind of an icon to show for a given filetype in a directory index.

You may wish to read the indexing tutorial for more information.

Syntax

```
AddIcon icon name1 name2...
```

`icon` is a virtual path to an image file which should be shown for files which match the pattern of `names`. Alternatively, this can be a group of the format (alt,icon) where alt is the text tag given for an icon for non–graphical browsers, and icon is a virtual path.

`name` is either `^^DIRECTORY^^` for directories (or `**DIRECTORY**` for backward compatibility), `^^BLANKICON^^`, specifying the blank icon used to format the list properly, a file extension (like .html), a partial filename, a wildcard expression, or a complete physical pathname.

You may use as many AddIcon directives as you wish.

Default

There are no default icons.

Example

```
AddIcon /icons/image.xbm .gif .jpg .xbm
```

When httpd is indexing and finds a file with the extension .gif, .jpg, or .xbm, it will reference `/icons/image.xbm` as an image to show next to the filename.

```
AddIcon /icons/dir.xbm ^^DIRECTORY^^
```

If a given directory entry is in fact a subdirectory, `/icons/dir.xbm` will be referenced as an image for that index entry.

```
AddIcon (SND,/icons/sound.xbm) *.au
```

This would reference `/icons/sound.xbm` as the image to show next to any sound file, with the textual ALT tag of SND for non–image clients.

- ## AddIconByType directive

  Purpose

  Tells httpd what kind of an icon to show for a given filetype in a directory index.

  You may wish to read the indexing tutorial for more information.

  Syntax

  `AddIconByType icon type1 type2...`

  `icon` is a virtual path to an image file which should be shown for files which match the pattern of `names`. Alternatively, this can be a group of the format (alt,icon) where alt is a 3–letter text tag given for an icon for non–graphical browsers, and icon is a virtual path.

  `name` is a wildcard expression of the MIME types for which to add this icon.

  You may use as many AddIconByType directives as you wish.

  Default

  There are no default icons.

  Example

  `AddIconByType /icons/image.xbm image/*`

  When httpd is indexing and finds a file which is an image, it will reference `/icons/image.xbm` as an image to show next to the filename.

  `AddIconByType (SND,/icons/sound.xbm) audio/*`

  This would reference `/icons/sound.xbm` as the image to show next to any sound file, with the textual ALT tag of SND for non–image clients.

- ## AddIconByEncoding directive

  Purpose

  Tells httpd what kind of an icon to show for a given filetype in a directory index.

  You may wish to read the indexing tutorial for more information.

  Syntax

```
AddIconByEncoding icon name1 name2...
```

`icon` is a virtual path to an image file which should be shown for files which match the pattern of `names`. Alternatively, this can be a group of the format (alt,icon) where alt is the text tag given for an icon for non–graphical browsers, and icon is a virtual path.

`name` is a wildcard expression specifying the content–encoding for which to display this icon.

You may use as many AddIconByEncoding directives as you wish.

Default

There are no default icons.

Example

```
AddIconByEncoding /icons/compress.xbm x-compress
```

When httpd is indexing, if it finds a file compressed with the UNIX compress command, it will display /icons/compress.xbm.

```
AddIconByEncoding (CMP,/icons/compress.xbm) x-compress
```

Same as above, but the ALT tag would be CMP for these items.

- ## IndexIgnore directive

Purpose

Tells httpd which files to ignore when generating an index of a directory.

Syntax

```
IndexIgnore pat1 pat2...
```

`pat` is a file extension or file name which should be ignored. When httpd is looking in a directory, it will try to match each of these strings to the right hand side of the entry's string, and if it matches it will ignore that entry in its directory index. If `pat` is a wildcard expression, httpd will match the filename against the given match expression.

Default

The only entry ignored by default is '.'.

Example

```
IndexIgnore README README.html .htaccess # ~
```

httpd will ignore files named README, README.html, and .htaccess when indexing a directory. It will also ignore emacs autosave files and emacs backup files.

- ## IndexOptions directive

  Purpose

  This directive specifies whether you want fancy directory indexing (with icons and file sizes) or standard directory indexing, and which options you want active for indexing.

  If your documents are being served from an AFS drive, you will want to seriously consider turning fancy indexing off.

  Syntax

  `IndexOptions opt1 opt2...`

  `opt` is an option name:

  - `FancyIndexing` turns FancyIndexing on
  - `IconsAreLinks` makes the icons part of the anchor for the filename
  - `ScanHTMLTitles` will cause httpd to fill in the description field of any unknown HTML document with its title (title must be within 256 bytes of the start of the file). You should NOT turn this option on unless your server has CPU time to spare.
  - `SuppressLastModified` will cause httpd not to print the last date of modification in index listings.
  - `SuppressSize` will cause httpd not to print the size of the files in index listings.
  - `SuppressDescription` will cause httpd not to print descriptions for any files.

  Only one IndexOptions directive should appear in the configuration file.

  Default

  If no IndexOptions directive is present, httpd assumes that none of the options should be on.

  Example

  `IndexOptions FancyIndexing IconsAreLinks`

  This will turn on fancy indexing with icons, and the icons will be part of the links to the files.

NCSA httpd match expressions

NCSA httpd version 1.2 or later supports wildcard expressions in various places in order to allow the administrator to specify patterns which should match strings.

Definition

The format of these expressions is very similar to shell wildcard patterns. The characters * and ? are allowed.

- `*` specifies 0 or more of any character.
- `?` specifies exactly one instance of any character.

Examples

`foo*` matches `foo`, `fooa`,and `foobar`, but does not match `afoo`.

`/*/.??*` matches any substring which contains /. followed by two characters. This pattern is useful for disabling UNIX hidden files such as /foo/bar/.htaccess.

NCSA httpd directory indexing

NCSA httpd provides a directory indexing format which is similar to that which will be offered by the WWW Common Library. To set up this indexing, follow these steps.

Activating Fancy indexing

You first need to tell httpd to use the advanced indexing instead of the simple version. The simple version should be used if you prefer its simplicity, or if you are serving files off of a remote file server, for which the stat() call would be costly. You tell the server which you want to use with either the IndexOptions directive, or the older FancyIndexing directive. We recommend:

```
IndexOptions FancyIndexing
```

Icons

NCSA httpd comes with a number of icons in the /icons subdirectory which are used for directory indexing. The first thing you should do is make sure your Server Resource Map has the following line in it:

```
Alias /icons/ /usr/local/etc/httpd/icons/
```

You should replace `/usr/local/etc/httpd/` with whatever you set ServerRoot to be.

Next, you need to tell the server what icons to provide for different types of files. You do this with the AddIcon and AddIconByType directives. We recommend something like the following setup:

```
AddIconByType (IMG,/icons/image.xbm) image/*
AddIconByType (SND,/icons/sound.xbm) audio/*
AddIconByType (TXT,/icons/text.xbm) text/*
```

This covers the three main types of files. If you want to add your own icons, simply create the appropriately sized xbm, place it in /icons, and choose a 3−letter ALT identifier for the type.

httpd also requires three special icons, one for directories, one which is a blank icon the same size as the other icons, and one which specifies the parent directory of this index. To use the icons in the distribution, use the following lines in srm.conf:

```
AddIcon /icons/menu.xbm ^^DIRECTORY^^
AddIcon /icons/blank.xbm ^^BLANKICON^^
AddIcon /icons/back.xbm ..
```

However, not all files fit one of these types. To provide a general icon for any unknown files, use the DefaultIcon directive:

```
DefaultIcon /icons/unknown.xbm
```

Descriptions

If you want to add descriptions to your files, use the AddDescription directive. For instance, to add the description "My pictures" to `/usr6/rob/public_html/images`, use the following line:

```
AddDescription "My pictures" /usr6/rob/public_html/images/*
```

If you want to have the titles of your HTML documents displayed for their descriptions, use the IndexOptions directive to activate `ScanHTMLTitles`:

```
IndexOptions FancyIndexing ScanHTMLTitles
```

WARNING: You should only use this option if your server has time to spare!!! This is a *costly* operation!

Ignoring certain items

Generally, you don't want httpd sending references to certain files when it's creating indexes. Such files are emacs autosave and backup files, httpd's .htaccess files, and perhaps any file beginning with . (if you have a gopher or FTP server running in that directory as well). We recommend you ignore the following patterns:

```
IndexIgnore */.??* */README* */HEADER*
```

This tells httpd to ignore any file beginning with ., and any file starting with README or HEADER.

Creating READMEs and HEADERs

When httpd is indexing a directory, it will look for two things and insert them into the index: A header, and a README. Generally, the header contains an HTML <H1> tag with a title for this index, and a brief description of what's in this directory. The README contains things you may want people to read about the items being served.

httpd will look for both plaintext and HTML versions of HEADERs or READMEs. If we add the following lines to srm.conf:

```
ReadmeName README
HeaderName HEADER
```

When httpd is indexing a directory, it will first look for HEADER.html. If it doesn't find that file, it will look for HEADER. If it finds neither, it will generate its own. If it finds one, it will insert it at the beginning of the index. Similarly, the server will look for README.html, then README to insert a trailer for the document.

Access Configuration

Overview

When creating a document tree, NCSA httpd allows you to control certain aspects of the branches (directories) of that tree.

- Access

  Restrict access to a branch to allowed hosts or authenticated users.

- Server features

  Branches considered unsafe (such as users' home directories) can be made more secure by disabling certain server functions in those directories.

Related tutorials

Please read this tutorial on setting up a secure server.

Marc has written a quick overview of setting up user authentication, which you may find helpful.

You will also want to know how to manage users for user authentication.

Methods

There are two methods for controlling access to directories.

- Global Access Configuration file

  A document in your server's `conf` directory, specified by the Server Configuration directive AccessConfig, controls access to any directory in your tree.

  httpd requires that you set up the Global ACF.

- Per−directory Access Configuration file

  Within your document tree, files with the name specified by the AccessFileName directive in the Resource Configuration File control access to the directory they are in as well as any subdirectories.

  Per−directory ACFs are optional. They can be restricted or completely forbidden by the Global ACF.

Directives

In addition to the general information which applies to all NCSA httpd configuration files, the access control files support the notion of sectioning directives.

Sectioning directives

Definition

Some access control directives require that certain information apply to all directives in a given section. These directives are called sectioning directives.

The formatting of these directives is similar to HTML tags.

For each sectioning directive, there must be two components: an opening directive, and a closing directive.

For instance, for the sectioning directive `Foo`, with one argument `data`, the opening directive would be:

```
<Foo data>
```

The closing argument would be:

```
</Foo>
```

The information given in the opening directive will affect all other directives between the opening and closing sectioning directive.

Example

The emphasized words are here for clarification; they would not appear in the actual configuration file.

```
<Directory /u/Web>                   The opening directive
require group physics
</Directory>                         The closing directive
```

`Directory` is the sectioning directive. Therefore, the information given by the opening directive, `/u/Web`, applies to the `require` directive within.

This is a list of the directives and sectioning directives used when writing an ACF. Each directive specifies where it can be used, and gives examples of usage.

- ## Directory directive

  Purpose

  Directory is a Sectioning directive which controls the directory to which access control directives apply.

  Scope

  This directive applies only to the Global ACF. All directives in the Global ACF must be contained in a Directory section.

  Syntax

  Opening Directive:

  `<Directory dir>`

  `dir` is the absolute pathname of the directory you are protecting. If you are using NCSA httpd 1.2 or later, this can also be a wildcard expression of a set of directories you wish to protect.

  Closing directive:

  `</Directory>`

  Example

  ```
  <Directory /u/Web>
  Options None
  </Directory>
  ```

  The directives contained within the Directory section above would only apply to the server directory /u/Web.

- ## Options directive

  Purpose

  The Options directive controls which server features are available in a given directory.

  Scope

  This directive applies to both the Global ACF as well as per−directory ACFs.

  Syntax

```
Options opt1 opt2 ... optn
```

Each `opt` is one of the following:

- None

  No features are enabled for in directory.

- All

  All features are enabled for in directory.

- FollowSymLinks

  The server will follow symbolic links in this directory.

- SymLinksIfOwnerMatch

  The server will only follow symbolic links for which the target file/directory is owned by the same user id as the link.

- ExecCGI

  Execution of CGI scripts is allowed in this directory.

- Includes

  Server side include files are enabled in this directory.

- Indexes

  The server allows users to request indexes in this directory. Disabling this option disables ONLY the server−generated indexes. It does not stop the server from sending any precompiled index file it may find in there (the name of which depends on DirectoryIndex.

- IncludesNoExec

  This enables server side includes in the directory, but disables the exec feature.

Default

If no Options directives are given for this directory or any of its parents, httpd assumes:

```
Options All
```

Example

```
Options Indexes FollowSymLinks
```

The server would allow users to index this directory and its subdirectories, and would allow symbolic links to be used within.

- ## AllowOverride directive

  Purpose

  The AllowOverride directive controls which access control directives can be overruled by a per–directory ACF.

  The global ACF cannot be restricted by this directive.

  Scope

  This directive may appear only in the global ACF.

  Syntax

  `AllowOverride or1 or2 ... orn`

  Each `or` is one of the following:

  - None

    ACFs are not allowed in this directory.

  - All

    ACFs are unrestricted in this directory.

  - Options

    Allow use of the Options directive.

  - FileInfo

    Allow use of the AddType and AddEncoding directives.

  - AuthConfig

    Allow use of these directives:
    - AuthName
    - AuthType
    - AuthUserFile
    - AuthGroupFile

  - Limit

    Allow use of the Limit sectioning directive.

  Default

  If no AllowOverride directives are given for this directory or any of its parents, httpd assumes:

`AllowOverride All`

Example

`AllowOverride Limit FileInfo`

ACFs in this directory are allowed to use `Limit`, `AddType`, and `AddEncoding`.

- ## AddType directive

Purpose

The AddType directive acts exactly as it does in the resource configuration file.

Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

Syntax

See the description from the resource configuration section.

- ## DefaultType directive

Purpose

The DefaultType directive acts exactly as it does in the resource configuration file.

Scope

This directive applies only to per−directory ACFs.

Syntax

See the description from the resource configuration section.

- ## AddEncoding directive

Purpose

The AddEncoding directive acts exactly as it does in the resource configuration file.

Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

Syntax

See the description from the resource configuration section.

- ## AddDescription directive

Purpose

The AddDescription directive acts exactly as it does in the resource configuration file.

Scope

This directive applies to only the per–directory ACFs.

Syntax

See the description from the resource configuration section.

- ## AddIcon directive

Purpose

The AddIcon directive acts exactly as it does in the resource configuration file.

Scope

This directive applies to both the Global ACF as well as per–directory ACFs.

Syntax

See the description from the resource configuration section.

- ## IndexIgnore directive

Purpose

The IndexIgnore directive acts exactly as it does in the resource configuration file.

Scope

This directive applies to both the Global ACF as well as per–directory ACFs.

Syntax

See the description from the resource configuration section.

- ## DefaultIcon directive

Purpose

The DefaultIcon directive acts exactly as it does in the resource configuration file.

Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

Syntax

See the description from the resource configuration section.

- ## ReadmeName directive

Purpose

The ReadmeName directive acts exactly as it does in the resource configuration file.

Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

Syntax

See the description from the resource configuration section.

- ## AuthName

Purpose

The AuthName directive sets the name of the authorization realm for this directory.
This realm is a name given to users so they know which username and password to send.

Scope

This directive applies to both global and per−directory ACFs.

This directive must be accompanied by AuthType, AuthUserFile, and AuthGroupFile directives in order for user authentication to work properly.

Syntax

```
AuthName name
```

Where name is a short name describing this authorization realm. Can contain spaces.

Default

There is no default.

Example

`AuthName PhysicsCollab`

Sets the authorization name of this directory to `PhysicsCollab`.

- ## AuthType

Purpose

The AuthType directive sets the type of authorization used in this directory.

Scope

This directive applies to both global and per−directory ACFs.

This directive must be accompanied by AuthName, AuthUserFile, and AuthGroupFile directives in order for user authentication to work properly.

Syntax

`AuthType type`

`type` is the authentication type to use for this directory. Only `Basic` is currently implemented.

Default

There is no default.

Example

`AuthType Basic`

Sets the authorization type of this directory to `Basic`.

- ## AuthUserFile

Purpose

The AuthUserFile directive sets the file to use as a list of users and passwords for user authentication.

Scope

This directive applies to both global and per−directory ACFs.

This directive must be accompanied by AuthName, AuthType, and AuthGroupFile directives in order for user authentication to work properly.

Syntax

`AuthUserFile path`

`path` is the absolute path of a user file created with the `htpasswd` support program.

Default

There is no default.

Example

`AuthUserFile /usr/local/etc/httpd/conf/.htpasswd`

Sets the authorization user file for this directory to `/usr/local/etc/httpd/conf/.htpasswd`.

- ## AuthGroupFile

Purpose

The AuthGroupFile directive sets the file to use as a list of user groups for user authentication.

Scope

This directive applies to both global and per−directory ACFs.

This directive must be accompanied by AuthName, AuthType, and AuthUserFile directives in order for user authentication to work properly.

Syntax

`AuthGroupFile path`

`path` is the absolute path of group file to use in this directory.

Default

There is no default.

Example

`AuthGroupFile /usr/local/etc/httpd/conf/.htgroup`

Sets the authorization group file for this directory to `/usr/local/etc/httpd/conf/.htgroup`.

* ## Limit directive

Purpose

Limit is a sectioning directive which controls which clients can access a directory.

Scope

This directive applies to both the global and per–directory ACFs.

Syntax

Opening Directive:

`<Limit meth1 meth2 ... methn>`

Each `meth` is one of the following methods:

  - `GET`

    Allows clients to retrieve documents and execute scripts.

  - `PUT`

    Not Implemented.

  - `POST`

    Not fully implemented. Currently, allows clients to use POST scripts.

Closing directive:

`</Limit>`

Only the following directives are allowed inside Limit sections:

  - **order**
  - **deny**
  - **allow**
  - **require**

A description of each will follow.

Example

```
<Limit GET>
order deny,allow
deny from all
```

```
allow from .ncsa.uiuc.edu
require group sdg
</Limit>
```

The only clients allowed to use the `GET` method in this directory must be from ncsa.uiuc.edu, and authenticate to group sdg.

---

order directive

Purpose

The order directive affects the order in which deny and allow directives are evaluated within a Limit section.

Scope

This directive is only available within Limit sections.

Syntax

`order ord`

`ord` is one of the following:

- `deny,allow`

  In this case, the deny directives are evaluated before the allow directives.

- `allow,deny`

  In this case, the allow directives are evaluated before the deny directives.

- `mutual-failure`

  This order is a bit unorthodox. With this order, you specify specific hosts which must be allowed or denied. Any host appearing on the allow list is allowed, and any list on the deny list is denied. Any appearing on neither is denied.

Default

If no order is given, httpd assumes:

`order deny,allow`

Example

```
<Limit /u/Web>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
</Limit>
```

In the /u/Web directory, the server evaluates the deny directive first. So, everyone is denied. It then evaluates the allow directive, and decides to allow clients from .ncsa.uiuc.edu.

---

deny directive

Purpose

The deny directive affects which hosts can access a given directory with a given method.

Scope

This directive is only available within Limit sections.

Syntax

`deny from host1 host2 ... hostn`

`host` is one of the following:

- A domain name

  A domain name, like .ncsa.uiuc.edu, which host names must end in to be allowed.

- A host name A full host name.

- A full IP address

  An IP address of a host.

- A partial IP address

  The first 1−3 bytes of an IP address, for subnet restriction.

- The keyword `all`

  This means that all hosts will be denied.

Default

No default applies.

Example

```
<Limit /u/Web>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
</Limit>
```

In the /u/Web directory, the server evaluates the deny directive first. So, everyone is denied. It then evaluates the allow directive, and decides to allow clients from .ncsa.uiuc.edu.

---

allow directive

Purpose

The allow directive affects which hosts can access a given directory with a given method.

Scope

This directive is only available within Limit sections.

Syntax

`allow from host1 host2 ... hostn`

`host` is one of the following:

- A domain name

  A domain name, like .ncsa.uiuc.edu, which host names must end in to be allowed.

- A host name A full host name.
- A full IP address

  An IP address of a host.

- A partial IP address

  The first 1−3 bytes of an IP address, for subnet restriction.

- The keyword `all`

  This means that all hosts will be allowed.

Default

No default applies.

Example

```
<Limit /u/Web>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
</Limit>
```

In the /u/Web directory, the server evaluates the deny directive first. So, everyone is denied. It then evaluates the allow directive, and decides to allow clients from .ncsa.uiuc.edu.

---

require directive

Purpose

The require directive affects which authenticated users can access a given directory with a given method.

Scope

This directive is only available within Limit sections.

Syntax

`require entity en1 en2 ... enn`

`en` are entity names, separated by spaces.

`entity` is one of the following:

- `user`

  Only the named users can access this directory with the given methods.

- `group`

  Only users in the named groups can access this directory with the given methods.

- `valid-user`

  All of the users defined in the AuthUserFile are allowed access upon providing a valid password.

Default

No default applies.

Example

```
<Limit GET PUT>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
require user ls
require group sdg
</Limit>
```

In this directory, the server evaluates the deny directive first. So, everyone is denied. It then evaluates the allow directive, and decides to allow clients from .ncsa.uiuc.edu. Now, it uses user authentication and only allows users who are named ls or are in the group sdg.

Example Configuration Files

For the security demo, I used the Global access configuration file which can be obtained from the same source from which you got this document.

The same effect could be achieved by taking everything within each Directorysection, and placing the directives in a ACF in each particular directory.

Script Selection

In the httpd distribution, you will find a directory called `cgi-bin`. It contains the default server scripts.

You do not need to keep any of the scripts. They are mostly intended as examples and models from which to build your own scripts.

Here is a brief list of the scripts and what they do to help you decide whether or not you wish to enable them.

- archie

  A very simple gateway to archie, using HTML <ISINDEX> tags.

- calendar

  Another HTML <ISINDEX> example, this one an interface to the UNIX cal command.

- date

  Prints the current date.

- uptime

  Prints the server's load average.

- finger

  A simple gateway to finger, using HTML <ISINDEX> tags.

- fortune

  Tell your fortune, mister?.

- jj

An HTML form to order a submarine sandwich.

- phf

  An HTML form interface to the PH system.

- query

  A general purpose form response script, intended to demonstrate basic form support concepts.

- imagemap

  Handles ISMAP queries etc. See the feature description and demo for what it does, and the setup document for how to use it.

Finding httpd a good home

At this stage, you should move httpd's control files from the directory you have been working in to the directory you defined as ServerRoot in the server configuration file.

Unless you have redefined their locations in the server configuration file, you will have to move the following files and directories into ServerRoot:

- `httpd` : The server itself
- `conf` : Configuration files
- `logs` : Access log and error log
- `support` : Support programs
- `cgi-bin` : Server scripts

The logs directory should be writable by the User your server is running as.

Starting Up

When executing httpd, you should remember the following command line flags:

- –d directory

  This specifies a different ServerRoot than /usr/local/etc/httpd and controls where httpd will look for its configuration files.

- –f file

  This specifies a Server Configuration File for httpd to start up with.

- –v

  If you ever get confused about which version of httpd you are using, this will print the software version.

Depending on your setting for ServerType, this step will be performed in two different ways:

Setting up under inetd

To set up inetd to recognize httpd and requests for it on a certain port, follow these instructions.

- Edit `/etc/services`

  Add a line which resembles the following to /etc/services:

  ```
  http          80/tcp
  ```

  You can replace 80 with whatever port number you want to serve from. The description of the port number (in the configuration docs above) may help if you're confused.

- Edit `/etc/inetd.conf`

  Add a line which resembles the following to /etc/inetd.conf:

  ```
  http       stream  tcp     nowait  nobody /usr/local/etc/httpd/httpd httpd<
  /PRE>

  Replace /usr/local/etc/httpd/httpd with wherever you are
  keeping the server binary, and nobody with the user name you want
  requests fulfilled as.
  ```

- ` Restart inetd`

  ```
  Find the inetd process using psand
  kill -HUP it to restart it.
  ```

Starting under Standalone

```
To start the server, simply execute the binary.

If you are using ServerRoot other
than /usr/local/etc/httpd, you will need to start the daemon with
httpd -d /your/server/root, where
/your/server/root is the setting of your ServerRoot
directive.

If you are installing as root, you will probably want to automatically
start httpd when the system comes up. This will be done through
modifications to various files in /etc/rc* depending on
your system. Consult your manuals for how to do this. Usually this is
done in /etc/rc.local.
```

```
If you have completed these steps, congratulations! Your server is
ready to go.

Before you run off to play with your new toy, please read
```

this guide to referencing files on
your server. Particularly, if you are wondering how to set up your
server's "home page", instructions are in there.

Constructing URLs to your server

Now that you have an http server, you'll want to reference it in your HTML documents and with your favorite Web browser.

Before you do anything, you should read this URL primer to familiarize yourself with URLs. You will want to pay attention to the section referring to HTTP URLs.

Definition

HTTP URLs have the basic form:

`http://servername:port/path`

servername

Your server's full hostname. It can be the server's real name or a DNS alias.

port

This is the port which your server is listening on. Specifying it in the URL is optional. If omitted, it is assumed to be 80.

If your ServerType is `inetd`, your port number was set in `/etc/services`.

If your ServerType is `standalone`, the Port directive set your port number.

path

This is the path to the document. *This is not the absolute pathname of the document on your machine.*

The server translates `path` as follows:

1. It looks for any defined Alias or ScriptAlias virtual names at the beginning of `path`. If it finds one, it replaces the virtual name with the real name and processes the request.

2. It looks for a prefix of /~, and if is not DISABLED, it will look in the user's public html subdirectory for the file.

3. It inserts DocumentRoot at the beginning of `path` and processes the request.

Setting up your Home Page

Some HTTP servers let you explicitly set up your home page (i.e. the page returned by the URL http://yourserver/).

To do this using NCSA httpd, create a DirectoryIndex file in the DocumentRoot directory.

Note that this index can be a symbolic link to another file.

Examples

My Resource Configuration file contains the following directives (among others):

```
DocumentRoot /u/Web
DirectoryIndex index.html
ScriptAlias /htbin /usr/local/etc/httpd/htbin
Alias /zftp /archive/ftp
```

An HTML document references http://hoohoo.ncsa.uiuc.edu/docs/Overview.html. The server finds no Alias or ScriptAlias virtual names in path, so it returns the file /u/Web/docs/Overview.html.

Someone references my home page as http://hoohoo.ncsa.uiuc.edu/. The server finds no virtual names, so it returns /u/Web/index.html.

Another HTML document references http://hoohoo.ncsa.uiuc.edu/htbin/uptime. The server finds the ScriptAlias /htbin at the beginning of path, and so executes the script /usr/local/etc/httpd/htbin/uptime.

Another HTML document references http://hoohoo.ncsa.uiuc.edu/zftp/README.txt. The server finds the Alias /zftp at the beginning of path, and returns the file /archive/ftp/README.txt.
