Last update: 25 Feb 2009

Grumpy web & gopher server's Manual

Written by Mateusz Viste

http://www.viste-family.net/mateusz/grumpy/

Table of Contents

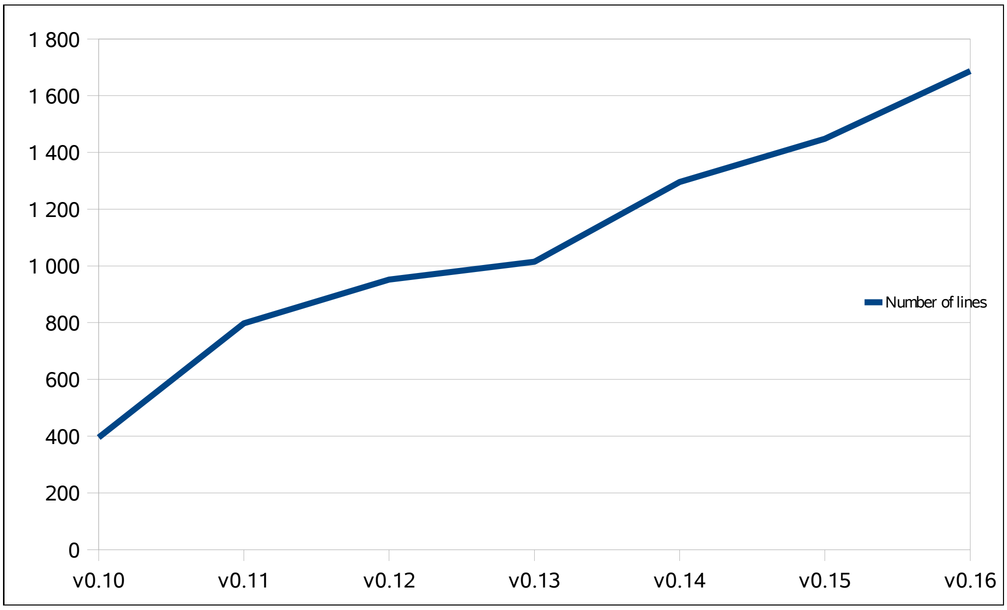
Introduction

Grumpy is a simple, easy to install, open-source web and gopher server for Linux. Grumpy is not a standalone daemon - it requires an inetd-compatible superserver to work.

Why one would prefer Grumpy over any other Linux web server, like Apache, Lighttp, Jigsaw...? Well, I wrote Grumpy primarily for fun, but it appears to have become a rather strong web server with some very good points: easy to install, lightweight, secure (as it doesn't support any dangerous features)...

The Grumpy web server is meant to be used for small projects, like home servers, some intranet applications, small-sized companies, etc. All the configuration is done via a single configuration file, which has very reasonable defaults. That makes Grumpy easily maintainable, and allows the administrator to have a full knowledge of what features are allowed/enabled on the server, and what's not. Grumpy supports CGI applications, may act as a gopher server, and is entirely written in FreeBASIC.

Grumpy is a small project, with very few developers (to tell you the truth, I'm the only one), but it keeps growing, and gets improved continuously. Below you will see a cool graph which presents the project's growth over the time.



...Yeah, I haven't anything better to put there. :-)

Installation (inetd)

Installing Grumpy on a Linux host is very easy. On the first place you will have to copy grumpy to /sbin or /usr/sbin, and grumpy.cfg to /etc. As mentioned previously, Grumpy needs an inetd-like superserver to work. If using the basic inetd, you will have to add the following line to /etc/inetd.conf:

```
www   stream      tcp   nowait      www-data    /sbin/grumpy      grumpy
```

Important things here are: "www", which is the name of service to bind to. You will have to check if the www service has its entry in /etc/services. "www-data" is the name of the user which has to be used to run the grumpy process. Never use root to run grumpy (or any other public daemon)! Obviously, you will have to check if the www-data user exists in your system (in Debian it exists by default), and let him write to /var/log/grumpy.log, read from /etc/grumpy.cfg and execute /sbin/grumpy. The easiest way to do that is simply to run "chown www-data file" on each file. When the inetd configuration is done, you'll have to restart the superserver (or the whole machine). In Debian, restarting inetd may be done by typing "kill -HUP `cat /var/run/inetd.pid`".

Don't forget to put an index.htm (or index.html) file into your RootDir path (by default, the RootDir is /var/www/). Note, that you don't need to use any index file if you allow to browse directories (AllowDirListing=1 in /etc/grumpy.cfg).

Installation (xinetd)

If your system is running xinetd, the first steps will be similar to the inetd case: on the first place you will have to copy grumpy to /sbin or /usr/sbin, and grumpy.cfg to /etc. The behavior of xinetd is very similar to inetd. However, the configuration file has a different syntax. Some distribution has an unique configuration file for xinetd, others uses separate files for each service (these files are usually in /etc/xinet.d/). In any case, you will need to add the following lines to let xinetd know about Grumpy:

```
service www
disable = no
socket_type = stream
protocol = tcp
wait = no
user = wwwrun
server = /sbin/grumpy
instances = 100
per_source = 10
log_type = FILE /var/log/xinetd-grumpy.log
log_on_success = HOST PID DURATION
log_on_failure = HOST
```

Important things here are: "service www", which is the name of service to bind to. You will have to check if the www service has its entry in /etc/services. "wwwrun" is the name of the user which has to be used to run the grumpy process. Never use root to run Grumpy (or any other public daemon)! However, you may want to use it for troubleshooting purpose. Obviously, you will have to check if the wwwrun user exists in your system, and let him write to /var/log/grumpy.log, read from /etc/grumpy.cfg and execute /sbin/grumpy. The easiest way to do that is simply to run "chown wwwrun file" on each file. When the xinetd configuration is done, you'll have to restart the superserver (or the whole machine). In most distributions, restarting xinetd may be done by running "/etc/init.d/xinetd restart". For more details on available features, see the xinetd manual.

Don't forget to put an index.htm (or index.html) file into your RootDir path (by default, the RootDir is /var/www/). Note, that you don't need to use any index file if you allow to browse directories (AllowDirListing=1 in /etc/grumpy.cfg).

Configuration file

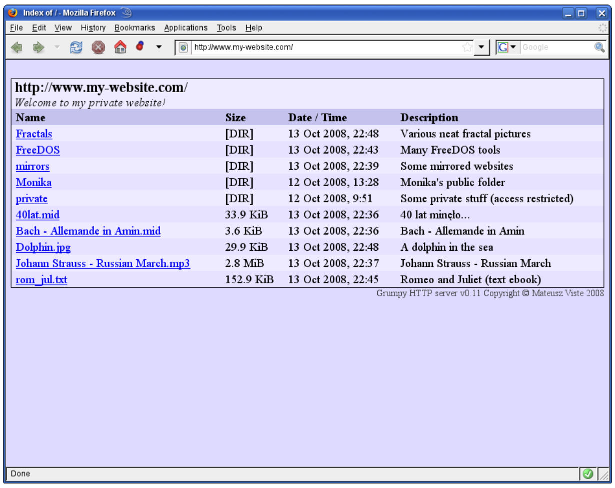
The Grumpy's configuration file should be at /etc/grumpy.cfg, and made readable by the user which will run Grumpy. It's a plain-text file, containing some tokens with values. Any line beginning by a "#" character is ignored. If, for some reasons, Grumpy would be unable to access/read his configuration file, he will take default values. Here's an example configuration file:

```
################################################
# Configuration file for the Grumpy web server #
################################################

## Set the loging verbosity ##
#  0 - No loging (not recommended, unless you really don’t care about logs)
#  1 - Log basic informations, like Request code/answer code. (Default)
#  2 - Maximum verbose (log full requests + a summary of the answer)
Verbose=1

## Allow/Disallow listing of directories ##
# If you allow directory listing, then Grumpy will list the content of
# directories which doesn’t have any index.htm or index.html file. You will
# probably want to enable it, unless you are paranoid.
# 0 - Disabled (default)
# 1 - Enabled
AllowDirListing=0

## The server’s root directory path. ##
# Default is /var/www
RootDir=/var/www

## QoS bandwith limitation ##
# This setting allows you to limit the transfer rate per file (in KiB/s).
# Eg. "QoS=5" means that Grumpy will serve files at a rate of max. 5 KiB/s.
# Note, that the QoS setting is set per file (an user could download one file
# at 5 KiB/s, two files at 10 KiB/s, etc...).
# 0 disables the QoS limitation (so Grumpy will serve files as fast as he can).
# Default is 0.
QoS=0

## Support for CGI applications. ##
# 0 - Disabled (default)
# 1 - Executes CGI, but only if the script has the *.cgi extension and is in
#&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;the /cgi-bin/ subdirectory
# 2 - Executes any file which have the *.cgi extension
#
# The CGI support in Grumpy is not standard. Read the Grumpy manual first.
# WARNING: AS ANY OTHER SERVER-SIDE CODE EXECUTION, THE CGI SUPPORT MAY
#          BE *VERY* DANGEROUS IF USED UNCORRECTLY. ENABLE IT ONLY IF YOU
#          KNOW EXACTLY WHAT YOU ARE DOING!
CgiSupport=0
```

```
## Virtual Hosts configuration ##
# Grumpy supports the use of virtual hosts. Any virtual host has to be
# declared below. The syntax is the following:
# vhost:my_hostname=my_directory
#
# Eg. vhost:mywebsite.com=/var/www/website/ would declare a virtual host
# called "mywebsite.com", and will redirect any requests to that host
# to the directory /var/www/website/.
#vhost:mywebsite.com=/var/www/webiste/

## Redirect domains ##
# Here you may declare any redirection for a given domain. It's particularily
# useful if you have moved your website to another domain name and wants to
# keep the content available for users (note, that crawling robots will follow
# such redirects, too). A domain redirect will generate a 301 HTTP response for
# any request destinated to the redirected domain.
# Syntax: Redirect:OldDomain=NewDomain
# Example:
# RedirectDomain:MyOldDomainName.net=www.MyNewFlashyDomainName.com

## Custom HTTP headers ##
# Below you may define few custom header. If set, these headers will be sent
# with any HTTP response to the client. If you leave them empty, they won't
# be sent.
# Example:
# x-powered-by=The Grumpy http &amp; gopher server, (C) Mateusz Viste
x-powered-by=
x-hosted-by=
x-server-admin=
x-disclaimer=

## Gopher configuration ##
# There comes the Gopher configuration. If you don't know what Gopher is,
# then don't bother about this section - you don't need it.
# The Gopher engine implemented into Grumpy needs to know three things:
# GopherRoot - that's the local path to Gopher ressources.
# GopherHostname - The public hostname the gopher server is available at.
# GopherPort - The port on which the public Gopher server listens on.
GopherRoot=/var/gopher/
GopherHostname=gopher.mydomain.net
GopherPort=70
```

Directory listing

Grumpy is able to list directories content on his own. That is, if there's no index.html nor index.htm file in the given directory, and you enabled directory listing in the configuration, then Grumpy will happily display an index, listing all files and subdirectories. Of course, if the directory requires authentication, Grumpy will ask for credentials first. Note, that the listing won't display any hidden ressource (ie file or directory beginning by a dot). It won't display the "descript.ion" file either.


Index of / - Mozilla Firefox
File Edit View History Bookmarks Applications Tools Help
http://www.my-website.com/
Google
Done

http://www.my-website.com/
Welcome to my private website!

Name	Size	Date / Time	Description
Fractals	[DIR]	13 Oct 2008, 22:48	Various neat fractal pictures
FreeDOS	[DIR]	13 Oct 2008, 22:43	Many FreeDOS tools
mirrors	[DIR]	13 Oct 2008, 22:39	Some mirrored websites
Monika	[DIR]	12 Oct 2008, 13:28	Monika's public folder
private	[DIR]	12 Oct 2008, 9:51	Some private stuff (access restricted)
40lat.mid	33.9 KiB	13 Oct 2008, 22:36	40 lat minęło...
Bach - Allemande in Amin.mid	3.6 KiB	13 Oct 2008, 22:36	Bach - Allemande in Amin
Dolphin.jpg	29.9 KiB	13 Oct 2008, 22:48	A dolphin in the sea
Johann Strauss - Russian March.mp3	2.8 MiB	13 Oct 2008, 22:37	Johann Strauss - Russian March
rom_jul.txt	152.9 KiB	13 Oct 2008, 22:45	Romeo and Juliet (text ebook)

Grumpy HTTP server v0.11 Copyright © Mateusz Viste 2008

I guess you are wondering now, what's that « descript.ion » file for? Well, it's a text file which contains any descriptions of the directory content. Grumpy will use it (if found) to fill the "Description" column of the listing. The structure of that file is very basic. Each entry has to be stored in one line: element[tab]description. It's very important to keep in mind that the delimiters are TABs, as it won't work with simple spaces (that's because you may have filenames containing spaces, so it would be confusing to allow spaces as delimiters). Here's a short example of a "descript.ion" file:

```
.     This directory contains some pictures of my dog
summer2008  Here are many photos from our summer 2008 holiday
ingrass.jpg Doggy in the grass
beach.jpg   Swimming dog
dream.jpg   Doggy felt asleep
```

The entry "." is describing the current directory (this description would appear in the bottom of the listing page). An important (well, maybe not THAT important, but rather "good to know") restriction of the "directory listing" feature is the fact, that Grumpy will display only the first 20'000 entries (not the first alphabetically, but rather the first in disk entries order). Therefore, if you have more than 20'000 elements (files and/or directories) in a single directory, you will probably prefer to build an index by yourself. Anyway, Grumpy will then display a big red warning stating that there are too many files to display.

HTTP Authentication

Basically, two authentication methods exist in HTTP/1.1: the "basic" method, where the username and password are transmitted in plaintext, and the "digest" method, where the username and password are replaced by their MD5 hash. Grumpy supports only the first one.

How does it work?

When a particular resource has been protected using basic authentication, Grumpy sends a 401 Authentication Required header with the response to the request, in order to notify the client that user credentials must be supplied in order for the resource to be returned as requested.

Upon receiving a 401 response header, the client's browser, if it supports basic authentication, will ask the user to supply a username and password to be sent to the server. If you are using a graphical browser, such as Netscape or Internet Explorer, what you will see is a box which pops up and gives you a place to type in your username and password, to be sent back to the server. If the username is in the approved list, and if the password supplied is correct, the resource will be returned to the client.

Because the HTTP protocol is stateless, each request will be treated in the same way, even though they are from the same client. That is, every resource which is requested from the server will have to supply authentication credentials over again in order to receive the resource. Fortunately, the browser takes care of the details here, so that you only have to type in your username and password one time per browser session - that is, you might have to type it in again the next time you open up your browser and visit the same web site.

Along with the 401 response, certain other information will be passed back to the client. In particular, it sends a name which is associated with the protected area of the web site. This is called the realm, or just the authentication name. The client browser caches the username and password that you supplied, and stores it along with the authentication realm, so that if other resources are requested from the same realm, the same username and password can be returned to authenticate that request without requiring the user to type them in again. This caching is usually just for the current browser session, but some browsers allow you to store them permanently, so that you never have to type in your password again.

How to set up authentication?

As mentioned before, all you have to do is put a ".grumpy.auth" file in the directory which access to has to be restricted. Such .grumpy.auth file would have the following structure (any line begining by a "#" character is ignored):

```
# Authentication file for Grumpy
realm=Password, please!
John:blahblah
Kim:white kitty
```

The realm token defines the realm (surprising, isn't it?). Usually the user's browser will display the realm in the popup window asking for credentials. Then, we have the list of users (with their passwords) which are allowed to access the given location. The syntax is very simple: "user:password". From the example above, we see that the users allowed to access the location are "John" and "Kim". John's password is "blahblah", while Kim's is "white kitty". Note, that the system is case-sensitive! A single ".grumpy.auth" file restricts the access to its own directory, as well as any subdirectories (unless these subdirectories have their own ".grumpy.auth" files). Of course, the ".grumpy.auth" file is not retrievable via HTTP. If someone request it, Grumpy will output a 404. Take care to not leave any (renamed) copy of the file ".grumpy.auth", as it would be readable by anyone!

Attention:

The password is passed from the client to the server in plain text across the network. Anyone listening somewhere in the packet's way with any variety of packet sniffer will be able to read the username and password in the clear as it goes across.

Not only that, but remember that the username and password are passed with every request, not just when the user first types them in. So the packet sniffer need not to be listening at a particularly strategic time, but just for long enough to see any single request come across the wire.

And, in addition to that, the content itself is also going across the network in the clear, and so if the web site contains sensitive information, the same packet sniffer would have access to that information as it went past, even if the username and password were not used to gain direct access to the web site.

CGI support

Grumpy supports CGI application, although it covers only a limited amount of functionalities. The only supported CGI programs are applications which output text data (binary data won't be interpreted properly by the Grumpy's CGI parser). Fortunately, CGI scripts outputting binary stuff are rather rare.

The CGI standard defines few ways of providing input data to the CGI application: the GET and POST methods. The GET method transfers data via URL parameters, thus the amount of data is limited (typically no more than 256 bytes). The POST method is more efficient, and transfers data as a response's payload. Grumpy supports only the GET method (on the other hand, it makes him more secure).

Let's see how does CGI work.

Each time a client requests the URL corresponding to your CGI program, the server will execute it in real-time. The output of your program will go more or less directly to the client.

The web server (in our case Grumpy), may provide some informations to the CGI application - either by launching it with some command-line parameters, or setting some environment variables. Basically, if the client's request doesn't contain any (non-encoded) "=" character in the URL, then all URL parameters will be used as command-line parameters of the CGI script. For example, a "GET /cgi-bin/myscript.cgi?param1&amp;param2" request will let Grumpy execute "myscript param1 param2". If any "=" character is found in the request, then no command-line parameters are provided. In any case, several environment variables may be set:

QUERY_STRING	The URL parameters, as provided by the client
SERVER_SOFTWARE	The name and version of the server software
SERVER_NAME	The server's hostname, DNS alias, or IP address, used for self-referencing links
GATEWAY_INTERFACE	The revision of the CGI specification, as supported by the server
HTTP_USER_AGENT	The client's user-agent
SCRIPT_NAME	Script name (for self-referencing links)
REQUEST_METHOD	The request method (GET, HEAD, POST...)
SERVER_PROTOCOL	The HTTP protocol version of the client's query
HTTP_VERSION	Same as SERVER_PROTOCOL
HTTP_COOKIE	The client's cookie
HTTP_REFERER	The HTTP referer
AUTH_USER	The authenticated user (if there was any authentication)
REMOTE_USER	Same as AUTH_USER
AUTH_TYPE	The authentication type used to authenticate the user (BASIC, DIGEST...)

When it comes to answer to the client, the CGI application will output a (partial) HTTP header, which should contain at least the "Content-Type" statement. This header will be followed by an empty line (which tells to the server that the header is over), and then will come the document which has to be transmitted to the client. Grumpy will analyze the header transmitted by the CGI application and will complete it with all lacking informations (the status code, HTTP version, etc).

As we just said, Grumpy is parsing the HTTP header returned by the CGI applications, and completes it when necessary. But there are situations where such behavior is unwanted (for example if the script undertakes to print the entire HTTP response including all necessary header fields.). That's what we call NPH scripts (NPH stands for "No Parsed Headers"). Grumpy is supporting such CGI applications - all we have to do, is name these particular programs starting with "nph-" (like "nph-myscript.cgi"). Grumpy is thereby instructed not to parse the headers (as it would normally do) nor add any which are missing. If you are going to use NPH, be sure to read and understand the HTTP spec (http://www.w3.org/Protocols/). Your headers should be complete and accurate, because you're instructing the Grumpy web server not to correct them or insert what's missing.

Here are some useful CGI references:

- The Common Gateway Interface <http://hoohoo.ncsa.uiuc.edu/cgi/>
- CGI Made Really Easy <http://www.jmarshall.com/easy/cgi/>
- CGI and Environment Variables <http://www.cs.cf.ac.uk/Dave/PERL/node200.html>

Customizing error pages

Grumpy does support customization of the returned error pages. The following error pages are customizable: 400, 401, 403, 404. By default, when one of these errors occurs, Grumpy returns a 4xx code with a html page containing a bare explanation of what happened. You can replace these defaults page by anything you like, by creating a directory called ".errors" in the root directory of your web server (or the root of your virtualhost server if you are using the virtualhost feature), and place one or several files named "400.htm", "401.htm", "403.htm" and "404.htm". If you're using virtual hosts, then each virtual host will use a distinct set of custom error pages.

Note, that if you are going to use any pictures or links on your custom error pages, you will need to use full (absolute) paths only (relative paths won't work).

Example

Say, you would like to change the default 404 error page to something nicer than just "Not found". We assume that your root path is /var/www/. You will have to make your custom 404 html page first, name it "404.htm" and put it into the directory /var/www/.errors/ (you will probably have to create the ".errors" directory before). That's all!

Gopher server

Yes, Grumpy features an embedded Gopher server, too!

Gopher is a distributed document search and retrieval network protocol designed for the Internet. Its goal is to function as an improved form of Anonymous FTP, enhanced with hyperlinking features similar to that of the World Wide Web.

Setting up the Gopher support in Grumpy is trivial. There are three settings to configure in the grumpy.cfg file:

```
GopherRoot=/var/gopher/
GopherHostname=gopher.mydomain.net
GopherPort=70
```

Then, you will have to set up your superserver (inetd or xinetd...) to listens on your Gopher port (in most cases you will want to use the port TCP/70), and bind it to /sbin/grumpy, adding the "--gopher" parameter to grumpy. A xinetd configuration file could look like that:

```
service gopher
{
      disable = no
      socket_type = stream
      protocol = tcp
      wait = no
      user = wwwrun
      server = /sbin/grumpy
      server_args = --gopher
      instances = 10
      per_source = 2
      log_type = FILE /var/log/xinetd-grumpy.log
      log_on_success = HOST PID DURATION
      log_on_failure = HOST
}
```

If you are using the classic (oldish) inetd, you will have to add a line to your inetd.conf configuration file similar to that one:

```
gopher   stream   tcp   nowait    wwwrun   /sbin/grumpy    grumpy --gopher
```

Do not forget to check if the service "gopher" is properly declared in your /etc/services file. Note, that Grumpy will look at any "descript.ion" files (if found in the browsed directory) when serving Gopher content. Besides that, the Gopher protocol needs to describe the type of any resource. Grumpy is simply basing on the file's extension to assign a Gopher type to the resource. Below is a table containing all relations between gopher filetype and real file's extension (at least that's the way Grumpy handles them):

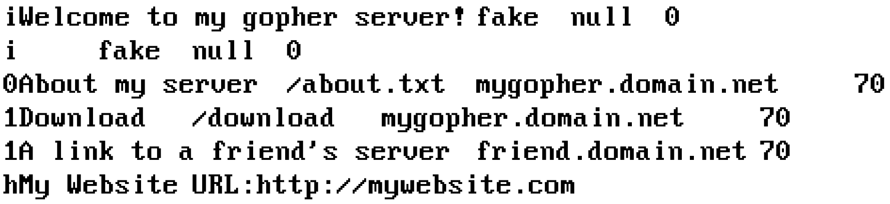
Gopher type	**Description**	**Files binded to this gopher type**
0	Plain text file	*.txt
1	Directory listing	All directories
2	CSO search query	-
3	Error message	-
4	BinHex encoded text file	-
5	Binary (PC-DOS) archive file	-
6	UUEncoded text file	-
7	Search engine query	-
8	Telnet session pointer	-
9	Binary file	All files which doesn't fit into any other category
g	GIF image file	*.gif
h	HTML file	*.htm, *.html, *.gopherlink containing an "URL" selector
i	Informational message	-
I	Image file (other than GIF)	*.jpg, *.jpeg, *.png, *.bmp, *.pcx, *.ico, *.tif, *.tiff, *.svg, *.eps
s	Audio file	*.mp3, *.mp2, *.wav, *.mid, *.wma, *.flac, *.mpc, *.aiff, *.aac
P	PDF file	*.pdf
M	MIME encoded message	-
;	Video file	-

The Gopher protocol is handling linking to external resources in a very neat way: links are part of the protocol itself, not part of the document. That's why creating gopher links requires a special trick. For the purpose of gopher links, Grumpy uses files with the extension *.gopherlink. Let's say, we would like to put a link to a gopher site located at gopher://mygopher.server.net/1/myfolder. We would create a file (say, "link-to-my-server.gopherlink") with the following content:

```
Server=mygopher.server.net
Selector=/myfolder
Type=1
Port=70
Description=This is a link to my folder on my gopher server
```

The only parameter of the file which is really required is (obviously) "Server". All other parameters will be set to default values (no selector, type=1, port=70). If no description is provided in the gopherlink file, then the server's address will be used. Note, that you may add links to HTTP servers, too. For a link pointing at http://www.mydomain.com/stuff.htm, you'll have to use a very short gopherlink file, containing just the "Selector" token: "Selector=URL:http://www.mydomain.com/stuff.htm", and optionally a description. In this case, the "Server" token is not used.

Last, but not least, there are situations where you would like to have the absolute control on what (and how) the server will display a directory. That's why Grumpy is supporting gophermaps. If Grumpy finds a gophermap in a directory, then it doesn't check the directory content, and simply outputs to the user the content of the gophermap. A gophermap must be named "grumpy.gophermap", and must contain gopher entries as described in the RFC 1436. There's an example of a "grumpy.gophermap" file (don't forget about TABs!):



```
iWelcome to my gopher server!	fake	null	0
i	fake	null	0
0About my server	/about.txt	mygopher.domain.net	70
1Download	/download	mygopher.domain.net	70
1A link to a friend's server	friend.domain.net	70
hMy Website	URL:http://mywebsite.com
```

Note, that you may omit the server's address and server's port parts. Grumpy will then use his general settings.

Legal mumbo-jumbo

Copyright © Mateusz Viste 2008, 2009

http://www.viste-family.net/mateusz/grumpy/

All rights reserved. This product or documentation is protected by copyright and is distributed under licenses restricting its use, copying, distribution and decompilation. See the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version for details.

The copyright owner gives no warranties and makes no representations about the contents of this manual and specifically disclaims warranties of merchantability or fitness to any purpose.

The copyright owner reserves the right to revise this manual and to make changes from time to time in its content without notifying any person of such revision or changes.

Trademarks

Unix is a registered trademark of UNIX System Laboratories, Inc. All other product names mentioned herein are the trademarks of their respective owners.