NAME

PyGopherd − Multiprotocol Information Server

SYNOPSIS

pygopherd [*configfile*]

DESCRIPTION

Welcome to **PyGopherd**. In a nutshell, **PyGopherd** is a modern dynamic multi-protocol hierarchical information server with a pluggable modularized extension system, full flexible caching, virtual files and folders, and autodetection of file types -- all with support for standardized yet extensible per-document metadata. Whew! Read on for information on this what all these buzzwords mean.

FEATURES

Here are some of **PyGopherd**'s features:

- Provides built-in support for multiple protocols: HTTP (Web), Gopher+, Gopher (RFC1436), Enhanced Gopher0, and WAP (mobile phones). Protocols can be enabled or disabled as desired.
- Provides protocol autodetection. That is, **PyGopherd** can listen for all the above protocols **on a single port** and will automatically respond using the protocol it detects the client is using. Practical effects of this are that you can, for instance, give out a single URL and have it viewable normally on desktop Web browsers and in WAP mode on mobile phones -- and appropriately in various Gopher browsers.
- Metadata and site links can be entered in a variety of formats, including full UMN dotfile metadata formats as well as Bucktooth gophermap files. Moreover, gophermap files are not limited to Gopher protocols, and can be used for all protocols.
- Support for inter-protocol linking (linking from Gopher sites to web sites)
- Virtual folder system lets you serve up anything as if it were regular files and directories. PyGopherd comes with the following virtual folder systems built in:
  - Can present any Unix MBOX, MMDF box, MH directory, Maildir directory, or Babyl mailbox as a virtual folder, the contents of which are the messages in the mailbox.
  - Can use a configurable separator to split a file into multiple parts, the first line of each becoming the name for the virtual folder.
  - Can peek inside a ZIP file and serve it up as first-class site citizens -- metadata can even be stored in the ZIP files.
  - Can serve up the contents of a dictd server as a filesystem.
- Modular, extensible design: you can use PyGopherd's own PYG extension format, or UMN- or Bucktooth-style executables.
- Runs on any platform supported by Python 2.2 or 2.3. This includes virtually every past and current flavor of Unix (Linux, *BSD, Solaris, SunOS), Windows, MacOS 9.x and X, and more. Some features may not be available on non-Unix platforms.
- Runs on any platform supported by Java 1.1 via the Jython Python implementation.
- Tunable server types via configuration directive -- forking or threading.
- Secure design with support for chrooted execution.
- Feature-complete, full implementations of: Gopher0 (RFC1435), Gopher+, HTTP, and WAP.
- Support for automatically finding the titles of HTML documents for presentation in a directory.
- Versatile configuration file format is both extensible and nicely complementary of the module system.
- Protocol-independant, handler-dependant caching. This increases performance by letting handlers cache dynamically-generated information -- currently used by the directory handlers. This can improve performance of directories by several orders of magnitude. Because this is a handler cache only, all protococls share the single cache. Since the processing time for the protocols is negligable, this works out

very well.

- Autosensing of MIME types and gopher0 item types. Both are completely configurable. MIME type detection is done using a standard mime.types file, and gopher0 types are calculated by using a configurable regexp-based MIME-to-gophertype map.
- Heavy support of regular expressions in configuration.
- ProtocolMultiplexer and HandlerMultiplexer let you choose only those protocols and handlers that you wish your server to support and the order in which they are tried when a request comes in.
- Full logging via syslog.

ABOUT GOPHER

PyGopherd started life as a server for the Gopher Internet protocol. With Gopher, you can mount a filesystem (viewing files and folders as if they were local), browse Gopherspace with a web browser, download files, and be interactive with searching.

But this is only part of the story. The world of Gopher is more expansive than this. There are two major gopher protocols: Gopher0 (also known as RFC1436) and Gopher+. Gopher0 is a small, simple, lightweight protocol that is very functional yet also extremely easy to implement. Gopher0 clients can be easily places in small embedded devices or in massive environments like a modern web browser.

Gopher+ is based on Gopher0 but extends it by providing document metadata such as file size and MIME type. Gopher+ allows all sorts of neat features, such as configurable metadata (serving up a bunch of photos? Add a Subject field to your metadata to let a customized photo browser display who is pictured) and multiple views of a file (let the user select to view your photos as PNG or JPEG).

QUICK START

If you have already installed **PyGopherd** system-wide, or your administrator has done that for you, your task for setting up **PyGopherd** for the first time is quite simple. You just need to set up your configuration file, make your folder directory, and run it!

You can quickly set up your configuration file. The distribution includes two files of interest: *conf/pygopherd.conf* and *conf/mime.types*. Debian users will find the configuration file pre-installed in */etc/pygopherd/pygopherd.conf* and the *mime.types* file provided by the system already.

Open up *pygopherd.conf* in your editor and adjust to suit. The file is heavily commented and you can refer to it for detailed information. Some settings to take a look at include: *detach*, *pidfile*, *port*, *usechroot*, *setuid*, *setgid*, and *root*. These may or may not work at their defaults for you. The remaining ones should be fine for a basic setup.

Invoke **PyGopherd** with **pygopherd path/to/configfile** (or **/etc/init.d/pygopherd start** on Debian). Place some files in the location specified by the *root* directive in the config file and you're ready to run!

INSTALLATION

If you are reading this document via the "man" command, it is likely that you have no installation tasks to perform; your system administra- tor has already installed **PyGopherd**. If you need to install it yourself, you have three options: a system-wide installation with Debian, system-wide installation with other systems, and a single-user installation. You can download the latest version of PyGopherd from <URL:http://quux.org/devel/gopher/pygopherd/>

DEBIAN SYSTEM-WIDE INSTALLATION

If you are tracking Debian unstable, you may install **PyGopherd** by simply running this command as root:

apt-get install pygopherd

If you are not tracking Debian unstable, download the .deb package from the **PyGopherd** website and then run **dpkg -i** to install the downloaded package. Then, skip to the configuration section below. You will use **/etc/init.d/pygopherd start** to start the program.

OTHER SYSTEM-WIDE INSTALLATION

Download the tar.gz version of the package from the website. Make sure you have Python 2.2 or above installed; if now, download and install it from <URL:http://www.python.org/>. Then run these commands:

tar -zxvf pygopherd-x.y.z.tar.gz
cd pygopherd-x.y.z
python2.2 setup.py

Some systems will use **python** or **python2.3** in place of **python2.2**.

Next, proceed to configuration. Make sure that the */etc/pygopherd/pygopherd.conf* file names valid users (*setuid* and *setgid* options) and a valid document root (*root* option).

You will type *pygopherd* to invoke the program.

SINGLE-ACCOUNT INSTALLATION

Download the tar.gz version of the package from the website. Make sure you have Python 2.2 or above installed; if now, download and install it from <URL:http://www.python.org/>. Then run these commands:

tar -zxvf pygopherd-z.y.z.tar.gz
cd pygopherd-x.y.z

Modify *conf/pygopherd.conf* as follows:

- Set *usechroot = no*
- Comment out (add a # sign to the start of the line) the *pidfile*, *setuid*, and *setgid* lines.
- Set *root* to osomething appropriate.
- Set *port* to a number greater than 1024.

When you want to run **PyGopherd**, you will issue the **cd** command as above and then type **PYTHONPATH=. bin/pygopherd**. There is no installation step necessary.

CONFIGURATION

PyGopherd is regulated by a configuratoin file normally stored in */etc/pygopherd/pygopherd.conf*. You can specify an alternate configuration file on the command line. The **PyGopherd** distribution ships with a sample *pygopherd.conf* file that thoroughly documents the configuration file options and settings.

OPTIONS

All **PyGopherd** configuratoin is done via the configuration file. Therefore, the program has only one command-line option:

configfile
: This option argument specifies the location of the configuration file that **PyGopherd** is to use.

HANDLERS

PyGopherd defines several handlers which are responsible for finding data on your server and presenting it to the user. The handlers are used to generate things like links to other documents and directory listings. They are also responsible for serving up regular files and even virtual folders.

Handlers are specified with the *handlers* option in *pygopherd.conf*. This option is a list of handlers to use. For each request that arrives, **PyGopherd** will ask each handler in turn whether or not it can handle the request, and will handle the request according to the first handler that is capable of doing so. If no handlers can handle the request, a file not found error is generated. See the example configuration file for an example.

The remaining parts of this section describe the different handlers that ship with **PyGopherd**.

dir.DirHandler

This handler is a basic one that generates menus based on the contents of a directory. It is used for directories that contain neither a *gophermap* file nor UMN-style links files, or situations where you have no need for either of those.

This handler simply reads the contents of your on-disk directory, determines the appropriate types of each file, and sends the result to the client. The descriptions of each item are usually set to the

filename, but the *html.HTMLFileTitleHandler* may override that.

gophermap.BuckGophermapHandler

This handler is used to generate directory listings based on *gophermap* files. It will not read the directory on-disk, instead serving content from the *gophermap* file only. Gophermaps are useful if you want to present a directory in which the files do not frequently change and there is general information to present. Overall, if you only wish to present information particular to certain files, you should consider using the abstract feature of UMN.UMNDirHandler.

The *gophermap* files contain two types of lines, which are described here using the same convention normally used for command line arguments. In this section, the symbol \t will be used to indicate a tab character, Control-I.

full line of informational text

gophertypeDESCRIPTION [\t*selector* **[** \t*host* **[** \t*port* **]]**]

Foo

EXAMPLES

Here are some example configurations for various situations. Please e-mail any other examples you have that may be useful to me.

MULTIPLE ACCOUNTS WITH MUTT

This example shows you how to set up **OfflineIMAP** to synchronize multiple accounts with the mutt mail reader.

Start by creating a directory to hold your folders by running **mkdir ~/Mail**. Then, in your *~/.offlineimaprc*, specify:

accounts = Personal, Work

Make sure that you have both an *[Account Personal]* and an *[Account Work]* section. The local repository for each account must have different *localfolder* path names. Also, make sure to enable *[mbnames]*.

In each local repository section, write something like this:

localfolders = ~/Mail/Personal

Finally, add these lines to your *~/.muttrc*:

```
source ~/path-to-mbnames-muttrc-mailboxes
folder-hook Personal set from="youremail@personal.com"
folder-hook Work set from="youremail@work.com"
set mbox_type=Maildir
set folder=$HOME/Mail
spoolfile=+Personal/INBOX
```

That's it!

UW-IMAPD AND REFERENCES

Some users with a UW-IMAPD server need to use **OfflineIMAP**'s "reference" feature to get at their mailboxes, specifying a reference of "~/Mail" or "#mh/" depending on the configuration. The below configuration from (originally from docwhat@gerf.org) shows using a *reference* of Mail, a *nametrans* that strips the leading Mail/ off incoming folder names, and a *folderfilter* that limits the folders synced to just three.

```
[Account Gerf]
localrepository = GerfLocal
```

```
remoterepository = GerfRemote

[Repository GerfLocal]
type = Maildir
localfolders = ~/Mail

[Repository GerfRemote]
type = IMAP
remotehost = gerf.org
ssl = yes
remoteuser = docwhat
reference = Mail
# Trims off the preceeding Mail on all the folder names.
nametrans = lambda foldername: \
        re.sub('^Mail/', '', foldername)
# Yeah, you have to mention the Mail dir, even though it
# would seem intuitive that reference would trim it.
folderfilter = lambda foldername: foldername in [
    'Mail/INBOX',
    'Mail/list/zaurus-general',
    'Mail/list/zaurus-dev',
    ]
maxconnections = 1
holdconnectionopen = no
```

PYTHONFILE CONFIGURATION FILE OPTION

You can have **OfflineIMAP** load up a Python file before evaluating the configuration file options that are Python expressions. This example is based on one supplied by Tommi Virtanen for this feature.

In *~/.offlineimap.rc*, he adds these options:

```
[general]
pythonfile=~/.offlineimap.py
[Repository foo]
foldersort=mycmp
```

Then, the *~/.offlineimap.py* file will contain:

```
prioritized = ['INBOX', 'personal', 'announce', 'list']

def mycmp(x, y):
  for prefix in prioritized:
    if x.startswith(prefix):
      return -1
    elif y.startswith(prefix):
      return +1
  return cmp(x, y)

def test_mycmp():
  import os, os.path
  folders=os.listdir(os.path.expanduser('~/data/mail/tv@hq.yok.utu.fi'))
  folders.sort(mycmp)
  print folders
```

This code snippet illustrates how the *foldersort* option can be customized with a Python function from the *pythonfile* to always synchronize certain folders first.

ERRORS

If you get one of some frequently-encountered or confusing errors, please check this section.

UID VALIDITY PROBLEM FOR FOLDER

IMAP servers use a unique ID (UID) to refer to a specific message. This number is guaranteed to be unique to a particular message **forever**. No other message in the same folder will ever get the same UID. UIDs are an integral part of **OfflineIMAP**'s synchronization scheme; they are used to match up messages on your computer to messages on the server.

Sometimes, the UIDs on the server might get reset. Usually this will happen if you delete and then recreate a folder. When you create a folder, the server will often start the UID back from 1. But **OfflineIMAP** might still have the UIDs from the previous folder by the same name stored. **OfflineIMAP** will detect this condition and skip the folder. This is GOOD, because it prevents data loss.

You can fix it by removing your local folder and cache data. For instance, if your folders are under *~/Folders* and the folder with the problem is INBOX, you'd type this:

rm -r ~/Folders/INBOX
rm -r ~/.offlineimap/Account-*AccountName*
rm -r ~/.offlineimap/Repository-*RepositoryName*

(Of course, replace AccountName and RepositoryName with the names as specified in *~/.offlineimaprc*).

Next time you run **OfflineIMAP**, it will re-download the folder with the new UIDs. Note that the procedure specified above will lose any local changes made to the folder.

Some IMAP servers are broken and do not support UIDs properly. If you continue to get this error for all your folders even after performing the above procedure, it is likely that your IMAP server falls into this category. **OfflineIMAP** is incompatible with such servers. Using **OfflineIMAP** with them will not destroy any mail, but at the same time, it will not actually synchronize it either. (**OfflineIMAP** will detect this condition and abort prior to synchronization.)

This question comes up frequently on the **OfflineIMAP** mailing list <URL:http://lists.complete.org/offlineimap@complete.org/>. You can find a detailed discussion <URL:http://lists.complete.org/offlineimap@complete.org/2003/04/msg00012.html.gz> of the problem there.

OTHER FREQUENTLY ASKED QUESTIONS

There are some other FAQs that might not fit into another section of the document, so they are discussed here.

What platforms does OfflineIMAP run on?

It should run on most platforms supported by Python, which are quite a few.

I'm using Mutt. Other IMAP sync programs require me to use "set maildir_trash=yes". Do I need to do that with OfflineIMAP?

No. **OfflineIMAP** is smart enough to figure out message deletion without this extra crutch. You'll get the best results if you don't use this setting, in fact.

I've upgraded and now OfflineIMAP crashes when I start it up! Why?

You need to upgrade your configuration file. See [XRef to UPGRADING.4.0] at the end of this manual.

How do I specify the names of my folders?

You do not need to. **OfflineIMAP** is smart enough to automatically figure out what folders are present on the IMAP server and synchronize them. You can use the *folderfilter* and *foldertrans* configuration file options to request certain folders and rename them as they come in if you like.

How can I prevent certain folders from being synced?

Use the *folderfilter* option in the configuration file.

How can I add or delete a folder?

OfflineIMAP does not currently provide this feature, but if you create a new folder on the IMAP server, it will be created locally automatically.

Are there any other warnings that I should be aware of?

Yes; see the Notes section below.

What is the mailbox name recorder (mbnames) for?

Some mail readers, such as Mutt, are not capable of automatically determining the names of your mailboxes. **OfflineIMAP** can help these programs by writing the names of the folders ni a format you specify. See the example *offlineimap.conf* for details.

Can I synchronize multiple accounts with OfflineIMAP?

Sure. Just name them all in the *accounts* line in the *general* section of the configuration file, and add a per-account section for each one.

Does OfflineIMAP support POP?

No. POP is not robust enough to do a completely reliable multi-machine synchronization like **OfflineIMAP** can do. **OfflineIMAP** will not support it.

Does OfflineIMAP support mailbox formats other than Maildir?

Not at present. There is no technical reason not to; just no demand yet. Maildir is a superior format anyway. However, **OfflineIMAP** can sync between two IMAP servers, and some IMAP servers support other formats. You could install an IMAP server on your local machine and have **OfflineIMAP** sync to that.

[technical] Why are your Maildir message filenames so huge?

OfflineIMAP has two relevant principles: 1) never modifying your messages in any way and 2) ensuring 100% reliable synchronizations. In order to do a reliable sync, **OfflineIMAP** must have a way to uniquely identify each e-mail. Three pieces of information are required to do this: your account name, the folder name, and the message UID. The account name can be calculated from the path in which your messages are. The folder name can usually be as well, BUT some mail clients move messages between folders by simply moving the file, leaving the name intact.

So, **OfflineIMAP** must store both a UID folder ID. The folder ID is necessary so **OfflineIMAP** can detect a message moved to a different folder. **OfflineIMAP** stores the UID (U= number) and an md5sum of the foldername (FMD5= number) to facilitate this.

What is the speed of OfflineIMAP's sync?

OfflineIMAP versions 2.0 and above contain a multithreaded system. A good way to experiment is by setting *maxsyncaccounts* to 3 and *maxconnections* to 3 in each account clause.

This lets OfflineIMAP open up multiple connections simultaneously. That will let it process multiple folders and messages at once. In most cases, this will increase performance of the sync.

Don't set the number too high. If you do that, things might actually slow down as your link gets saturated. Also, too many connections can cause mail servers to have excessive load. Administrators might take unkindly to this, and the server might bog down. There are many variables in the optimal setting; experimentation may help.

An informal benchmark yields these results for my setup:

- 10 minutes with MacOS X Mail.app "manual cache"
- 5 minutes with GNUS agent sync
- 20 seconds with OfflineIMAP 1.x
- 9 seconds with OfflineIMAP 2.x
- 3 seconds with OfflineIMAP 3.x "cold start"
- 2 seconds with OfflineIMAP 3.x "held connection"

CONFORMING TO

- Internet Message Access Protocol version 4rev1 (IMAP 4rev1) as specified in RFC2060 and RFC3501
- CRAM-MD5 as specified in RFC2195
- Maildir as specified in the Maildir manpage <URL:http://www.qmail.org/qmail-manual-html/man5/maildir.html> and the qmail website <URL:http://cr.yp.to/proto/maildir.html>.
- Standard Python 2.2.1 as implemented on POSIX-compliant systems.

NOTES

DELETING LOCAL FOLDERS

OfflineIMAP does a two-way synchronization. That is, if you make a change to the mail on the server, it will be propogated to your local copy, and vise-versa. Some people might think that it would be wise to just delete all their local mail folders periodically. If you do this with **OfflineIMAP**, remember to also remove your local status cache (*~/.offlineimap* by default). Otherwise, **OfflineIMAP** will take this as an intentional deletion of many messages and will interpret your action as requesting them to be deleted from the server as well. (If you don't understand this, don't worry; you probably won't encounter this situation)

MULTIPLE INSTANCES

OfflineIMAP is not designed to have several instances (for instance, a cron job and an interactive invocation) run over the same mailbox simultaneously. It will perform a check on startup and abort if another **OfflineIMAP** is already running. If you need to schedule synchronizations, please use the *autorefresh* settings rather than cron. Alternatively, you can set a separate *metadata* directory for each instance.

COPYING MESSAGES BETWEEN FOLDERS

Normally, when you copy a message between folders or add a new message to a folder locally, **OfflineIMAP** will just do the right thing. However, sometimes this can be tricky -- if your IMAP server does not provide the SEARCH command, or does not return something useful, **OfflineIMAP** cannot determine the new UID of the message. So, in these rare instances, OfflineIMAP will upload the message to the IMAP server and delete it from your local folder. Then, on your next sync, the message will be re-downloaded with the proper UID. **OfflineIMAP** makes sure that the message was properly uploaded before deleting it, so there should be no risk of data loss.

USE WITH EVOLUTION

OfflineIMAP can work with Evolution. To do so, first configure your **OfflineIMAP** account to have **sep = /** in its configuration. Then, configure Evolution with the "Maildir-format mail directories" server type. For the path, you will need to specify the name of the top-level folder **inside** your **OfflineIMAP** storage location. You're now set!

USE WITH KMAIL

At this time, I believe that **OfflineIMAP** with Maildirs is not compatible with KMail. KMail cannot work in any mode other than to move all messages out of all folders immediately, which (besides being annoying and fundamentally broken) is incompatible with **OfflineIMAP**.

However, I have made KMail version 3 work well with **OfflineIMAP** by installing an IMAP server on my local machine, having **OfflineIMAP** sync to that, and pointing KMail at the same server.

MAILING LIST

There is an OfflineIMAP mailing list available. To subscribe, send the text "Subscribe" in the subject of a mail to offlineimap-request@complete.org. To post, send the message to offlineimap@complete.org. Archives are available at
<URL:http://lists.complete.org/offlineimap@complete.org/>.

BUGS

Reports of bugs should be sent via e-mail to the **OfflineIMAP** bug-tracking system (BTS) at offlineimap@bugs.complete.org or submitted online using the web interface <URL:http://bugs.complete.org/>.

The Web site also lists all current bugs, where you can check their status or contribute to fixing them.

UPGRADING TO 4.0

If you are upgrading from a version of **OfflineIMAP** prior to 3.99.12, you will find that you will get errors when **OfflineIMAP** starts up (relating to ConfigParser or AccountHashGenerator) and the configuration file. This is because the config file format had to change to accomodate new features in 4.0. Fortunately, it's not difficult to adjust it to suit.

First thing you need to do is stop any running **OfflineIMAP** instance, making sure first that it's synced all your mail. Then, modify your ~*/.offlineimaprc* file. You'll need to split up each account section (make sure that it now starts with "Account ") into two Repository sections (one for the local side and another for the remote side.) See the files *offlineimap.conf.minimal* and *offlineimap.conf* in the distribution if you need more assistance.

OfflineIMAP's status directory area has also changed. Therefore, you should delete everything in ~ /.offlineimap as well as your local mail folders.

When you start up **OfflineIMAP** 4.0, it will re-download all your mail from the server and then you can continue using it like normal.

COPYRIGHT

AUTHOR

OfflineIMAP, its libraries, documentation, and all included files, except where noted, was written by John Goerzen <jgoerzen@complete.org> and copyright is held as stated in the COPYRIGHT section.

OfflineIMAP may be downloaded, and information found, from its homepage via either Gopher <URL:gopher://quux.org/1/devel/offlineimap> or HTTP <URL:http://quux.org/devel/offlineimap>.

OfflineIMAP may also be downloaded using Subversion. Additionally, the distributed tar.gz may be updated with a simple "svn update" command; it is ready to go. For information on getting OfflineIMAP with Subversion, please visit the complete.org Subversion page <URL:http://svn.complete.org/>.

SEE ALSO

mutt(1), **python**(1)

HISTORY

Detailed history may be found in the file ChangeLog in the **OfflineIMAP** distribution. Feature and bug histories may be found in the file debian/changelog which, despite its name, is not really Debian-specific. This section provides a large overview.

Development on **OfflineIMAP** began on June 18, 2002. Version 1.0.0 was released three days later on June 21, 2002. Point releases followed, including speed optimizations and some compatibility fixes.

Version 2.0.0 was released on July 3, 2002, and represented the first time the synchronization became multithreaded and, to the best of my knowledge, the first multithreaded IMAP syncrhonizing application in existance. The last 2.0.x release, 2.0.8, was made on July 9.

Version 3.0.0 was released on July 11, 2002, and introduced modular user interfaces and the first GUI interface for **OfflineIMAP**. This manual also was introduced with 3.0.0, along with many command-line options. Version 3.1.0 was released on July 21, adding the Noninteractive user interfaces, profiling support,

and several bugfixes. 3.2.0 was released on July 24, adding support for the Blinkenlights GUI interface. **OfflineIMAP** entered maintenance mode for awhile, as it had reached a feature-complete milestone in my mind.

The 3.99.x branch began in on October 7, 2002, to begin work for 4.0. The Curses.Blinkenlights interface was added in 3.99.6, and many architectural changes were made.

4.0.0 was released on July 18, 2003, including the ability to synchronize directly between two IMAP servers, the first re-architecting of the configuration file to refine the notion of an account, and the new Curses interface.