Gaim play

# CHAT COLLECTOR

Are you interested in storing, organizing, and searching instant messaging conversations on your IMAP server? The Perl script in this month's column can help you do just that. **BY MIKE SCHILLI**

Storing emails on an IMAP server instead of on a local PC has the advantage of providing access to information you need, no matter where you are when you need it. But if you use Instant Messaging in addition to email, the exchange of information is lost as soon as you finish chatting.

Many messaging clients, such as the ubiquitous Gaim, have a logging feature to help you solve this problem. The client records your messages on disk, but often you'll find yourself miles away from home when you need that vital URL that a friend IMed you recently.

## Security Risks

Of course, there is nothing to stop you from saving your logfiles on a server with public access and adding all kinds of search functions into the bargain. However, this raises the problem of protecting your data against unauthorized access. Although no one in his right mind would exchange confidential information over insecure chat channels, it is important to keep private conversations secure. If your new server happens to have a security hole, it would be embarrassing to see private chats exposed.

Because there is a tried and trusted, and relatively secure, place to store email – the IMAP server – it makes sense to deposit the log files from your message client there.

## Taking Minutes

To tell Gaim to log all conversations, you can easily use the Preferences *Logging* menu. I opted for the *Plain* format (see Figure 1) because I'm a dinosaur and still use Pine as my email client. I actually dislike HTML emails because of their inherent security problems and prefer plain text. The radio buttons *Log all instant messages* and *Log all chats* control logging of normal conversations and group chats. When you enable these features, Gaim automatically creates a separate text file for each conversation in *~/.gaim/logs*. Gaim 1.x versions use a path of *Provider/Sender/Receiver/*.txt* for this, so if the local user *mikeschilli* used the Yahoo Messenger protocol just before 10:00 on March 28, 2007, to talk to *randomperlhacker*, the local file would be stored as *~/.gaim/logs/yahoo/mikeschilli/randomperlhacker/2007-03-28.095243.txt*. In Figure 2, you can see the conversation and, in Figure 3, the logfile content.

## Chat Subjects

The *gaim2imap* daemon (Listing 1) calls the *update()* function to process any new logfiles it has found and then goes

back to sleep for the preset time interval. A setting of one hour (3600 seconds) is defined for this in the *$sleep* variable.

Instead of sending unprocessed logfiles as individual email messages to the IMAP server, the daemon first adds some meta information. The sender (*From:*) is set to the name of the other party, and a pseudo-domain of *@gaim* is added to prevent the IMAP server and the email client you use to read the message later from complaining. The email date is set to the start point of the conversation and formatted to comply with RFC822 by the *DateTime::Format::Mail* module. It would be helpful to show the most important topics of the conversation in the subject line of the email. For the chat in Figure 2, these topics could be "characters, perl, word, split, know, bit". Of course, topic extraction is a science unto itself, but *gaim2imap* is quite happy with just a couple of simple tricks, and this helps you achieve usable results.

## Stopwords

First of all, the *chat_process()* function attempts to identify the predominant language in the conversation. If you talk to international partners, you might use a mix of English, Spanish, or some other languages. The *Text::Language::Guess* CPAN module makes fairly intelligent guesses if the options are restricted to just two or three languages.

Then *chat_process()* attempts to identify stopwords [2] in the text; these are words that don't really tell much about the topic of the conversation, but are crucial to understanding the language. Articles (a, the), personal pronouns (I, you, he), or conjunctions (and, or) are a few examples of stopwords. For example, if a search engine receives a query such as *By the way, where is San Francisco?*, it will just ditch everything apart from the name of the city, and look for San Francisco in its index.

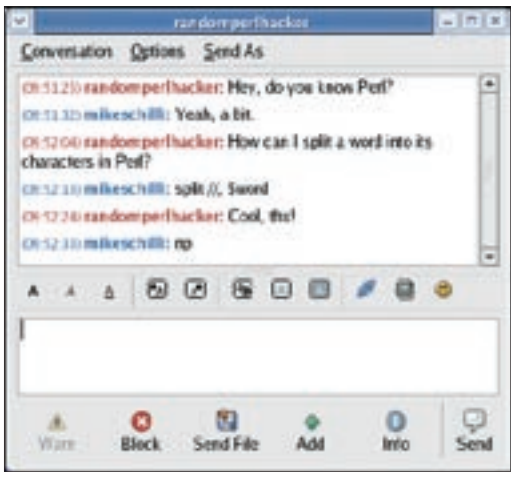


**Figure 2: A conversation with Gaim ...**

The *gaim2imap* script eliminates stopwords via the *Lingua::StopWords* CPAN module. Just specify a language, and then the module returns a reference to a hash whose keys are a collection of all the stopwords known to the module for this particular language. The script additionally defines a list of frequently encountered words from the list in *$im_stopwords* in line 27; you can normally assume that these words will not contribute much to the content.

To filter out the most important topics, the script uses a kind of home-grown approach: It counts how often specific words occur, weights them by frequency, and adds three extra points to long words (with more than six letters). If you like, you can integrate a more sophisticated approach; Yahoo, my employer, offers a Web API [3] for extracting topics from English texts.

## Lighting up Links

If an IM logfile contains one or multiple URLs, it might prove particularly valuable. *gaim2imap* uses the *URI::Find* CPAN module to discover URLs in the clear text of the chat. The constructor expects a callback function as an argument that gets called for every URL found. In *gaim2imap*, the callback function returns an empty string to have URLs eliminated from the text before it goes to the term-extraction stage. If the number of identified URLs is greater than 0, a **L** (for *Link*) is added at the start of the future email's subject line. With this in place, you can quickly see within your email client which logfiles, from a list of several, contain critical links. To make sure that the logs are as easy to read as emails, the body text of individual chat messages is formatted to a line length of 70 characters and justified with the *Text::Wrap* module and its *fill()* function in line 150. The script modifies *$Text::Wrap::columns* to accomplish this.

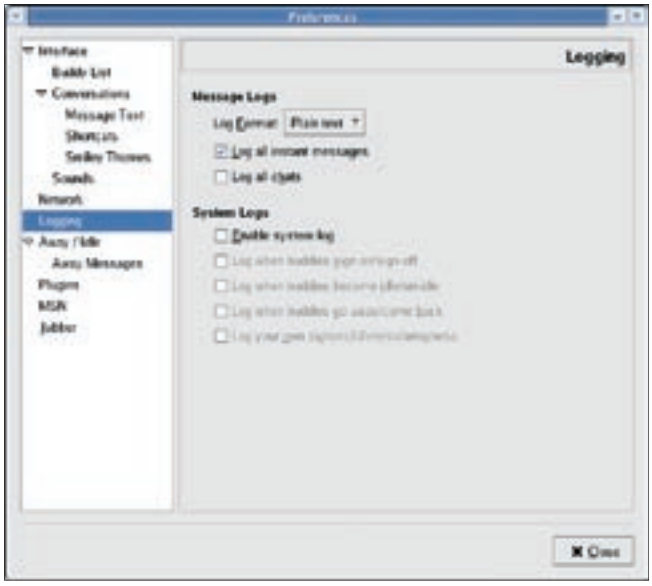


**Figure 1: Gaim's Preferences menu lets you configure how you log chat sessions.**

The *chat_process()* function returns a total of three values: the suggested subject line for the outgoing mail, the newly formatted text, and the start point of the chat session in Unix seconds.

The *imap_add()* function in line 220 creates a mail header from this and uses the *IMAP::Client* CPAN module's *append()* method to drop the message into the IMAP server's *im_mailbox* folder. Check out the Installation section to discover how to set up this folder on the IMAP server.

## Connecting

*IMAP::Client* is first switched to *RaiseError* mode by calling *onfail('ABORT')*. Any errors that occur will throw an exception in this mode, immediately causing the script to quit. This way, you don't need to check the return values of the individual methods the module offers. If you prefer the daemon not to quit, you can use *eval* to catch the exceptions and react to them.

The *connect()* method in line 77 establishes the connection to the IMAP server. In this script, *localhost* is the server

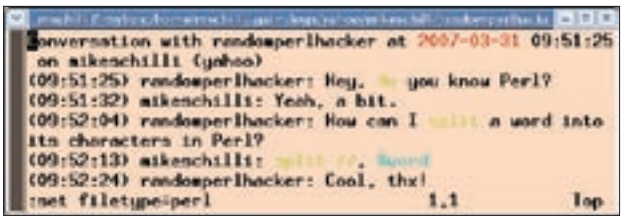


**Figure 3: ... and the matching logfile.**

because the Dovecot IMAP server is running on my Linux desktop. But *connect()* could just as easily contact any host on the Internet. Line 83 calls *authenticate()* and passes in the username and the password for the Unix user. The latter is collected by *gaim2imap* at program startup. Password input is handled by the *password_read()* function in line 42, and it comes from the bottomless treasure trove of the *Sysadm::Install* CPAN module.

The *Gaim::Log::Finder* and *Gaim::Log::Parser* CPAN modules find and parse the Gaim logs, removing the need to manually traverse directory hierarchies and extract dates and messages from each file encountered. The CPAN modules provide methods for getting sender and receiver information, dates, and the content of a chat session. The *chat_process()* function uses *$parser->next_message()* to iterate over all exchanged messages of a logfile, each returned as a *Gaim::Log::Message* object. The object features *date()*, *from()*, *to()*,

## Listing 1: gaim2imap

```
001 #!/usr/bin/perl -w
002 ##################################
003 # gaim2imap - IMAP chat logs
004 # Mike Schilli, 2007
005 # (m@perlmeister.com)
006 ##################################
007 use strict;
008 use Gaim::Log::Parser 0.04;
009 use Gaim::Log::Finder;
010 use Sysadm::Install 0.23
011   qw(:all);
012 use Lingua::StopWords;
013 use Text::Language::Guess;
014 use Log::Log4perl qw(:easy);
015 use Text::Wrap
016   qw(fill $columns);
017 use URI::Find;
018 use IMAP::Client;
019 use DateTime::Format::Mail;
020
021 my $mailbox = "im_mailbox";
022 my $tzone   =
023   "America/Los_Angeles";
024 my $min_age = 3600;
025 my $sleep   = 3600;
026
027 my $im_stopwords =
028   map { $_ => 1 } qw(maybe
029  thanks thx doesn hey put
030  already said say would can
031  could haha hehe see well
032  think like heh now many lol
033  doh );
034
035 Log::Log4perl->easy_init({
036   level    => $DEBUG,
037   category => "main",
038   file     =>
039 ">>$ENV{HOME}/.gaim2imap.log"
040 });
041
042 my $PW = password_read(
043           "password: ");
044
045 my $pid = fork();
046 die "fork failed"
047   if !defined $pid;
048 exit 0 if $pid;
049
050 dbmopen my %SEEN,
051   "$ENV{HOME}/.gaim/.seen",
052   0644
053   or LOGDIE
054 "Cannot open dbm file ($!)";
055
056 $SIG{TERM} = sub {
057  INFO "Exiting";
058  dbmclose %SEEN;
059  exit 0;
060 };
061
062 while (1) {
063  update();
064  INFO "Sleeping $sleep secs";
065  sleep $sleep;
066 }
067
068 ##################################
069 sub update {
070 ##################################
071  DEBUG "Connecting to IMAP";
072
073  my $imap =
074        IMAP::Client()->new();
075
076  $imap->onfail('ABORT');
077  $imap->connect(
078   PeerAddr => 'localhost',
079   ConnectMethod => 'PLAIN'
080  );
081
082  my $u = getpwuid $>;
083  $imap->authenticate($u,$PW);
084
085  my $finder =
086    Gaim::Log::Finder->new(
087   callback => sub {
088    my ($self, $file,
089        $protocol, $from, $to)
090      = @_;
091
092    return 1 if $from eq $to;
093
094    my $mtime =
095      (stat $file)[9];
096    my $age = time() - $mtime;
097
098    return 1
099      if $SEEN{$file}
100      and $SEEN{$file} ==
101      $mtime;
102
103    if ($age < $min_age) {
104     INFO "$file: Too ",
105          " recent ($age)";
106     return 1;
107    }
108
109    $SEEN{$file} = $mtime;
110    INFO "Processing log ",
111         "file: $file";
112    my ($subject, $formatted,
113        $epoch)
114      = chat_process($file);
115
116    imap_add( $imap,
117     $mailbox, $epoch,
118     "$to\@gaim", "",
119     $subject, $formatted );
120   });
121
122  $finder->find();
123 }
124
125 ##################################
126 sub chat_process {
```

and *content()* methods for accessing the date, the conversation partners, and the text of each message, respectively.

To tell *gaim2imap* to disappear into the background when the user enters the password at startup, line 45 *fork*s a child process. The parent process disappears without much ado, and the user sees the command-line's prompt return. The child process just goes on running. If the admin later kills the daemon by calling *kill* with the appropriate process ID, *gaim2imap* will try to save a persistent hash with seen log files with *dbmclose* before calling *exit* and quitting. The global *%SIG* hash uses a *$SIG{TERM}* entry to define this behavior.

An IM session could go on for several hours, but Gaim will keep on adding messages to an existing logfile. Gaim will not create a new file until the communication dialog is closed and a new message exchange starts.

The daemon that generates emails from the logfiles defines an hour of inactivity as the threshold. After this, the file is converted to an email and tagged as processed. If the session was still active at this time and if Gaim later appends a new message, the daemon will notice the change; the daemon stores the last modification time for each file it processes and stores this value in *%SEEN*, a persistent hash that is linked to a file by *dbmopen*. To avoid loss of data in this fairly rare scenario, the daemon simply reprocesses the file, producing a duplicate rather than giving up on potentially valuable content.

## Listing 1: gaim2imap

```
127 ################################
128  my ($file) = @_;
129
130  my $parser =
131    Gaim::Log::Parser->new(
132      file => $file);
133
134    # Search+delete URLs
135  my $urifind =
136   URI::Find->new(sub { "" });
137
138  my $text      = "";
139  my $formatted = "";
140  my $urifound;
141  $Text::Wrap::columns = 70;
142
143  while (my $m =
144   $parser->next_message()) {
145
146   my $content =
147     $m->content();
148   $content =~ s/\n+/ /g;
149
150   $formatted .= fill(
151    "", "  ",
152    nice_time($m->date())
153      . " "
154      . $m->from() . ": "
155      . $content
156     ) . "\n\n";
157
158   $urifound =
159     $urifind->find(
160    \$content);
161   $text .= " " . $content;
162  }
163
164  my $guesser =
165   Text::Language::Guess->new(
166   languages => [ 'en', 'es' ]
167  );
168
169  my $lang = $guesser
170    ->language_guess_string(
171                       $text);
172
173  $lang = 'en' unless $lang;
174
175  DEBUG "Guessed: $lang\n";
176
177  my $stopwords =
178  Lingua::StopWords::
    getStopWords(
179            $lang);
180
181  my %words;
182
183  while (
184   $text =~ /\b(\w+)\b/g) {
185
186   my $word = lc($1);
187   next
188     if $stopwords->{$word};
189   next if $word =~ /^\d+$/;
190   next if length($word) <= 2;
191   next if exists
192     $im_stopwords{$word};
193   $words{$word}++;
194   $words{$word} += 3
195     if length $word > 6;
196  }
197
198  my @weighted_words = sort {
199   $words{$b} <=> $words{$a}
200  } keys %words;
201
202  my $subj =
203    ($urifound ? '*L*' : "");
204  my $char = "";
205
206  while ( @weighted_words
207   and length($subj) + length(
208    $char . $weighted_words[0]
209   ) <= 70) {
210    $subj .= $char
211      . shift @weighted_words;
212    $char = ", ";
213   }
214
215  return ($subj, $formatted,
216   $parser->{dt}->epoch());
217 }
218
219 ################################
220 sub imap_add {
221 ################################
222  my ( $imap, $mailbox,
223       $date, $from, $to,
224       $subject, $text ) = @_;
225
226  $date = DateTime::Format::
    Mail
227    ->format_datetime(
228   DateTime->from_epoch(
229    epoch     => $date,
230    time_zone => $tzone
231   ));
232
233  my $message =
234      "Date: $date\n"
235    . "From: $from\n"
236    . "To: $to\n"
237    . "Subject: "
238    . "$subject\n\n$text";
239
240  my $fl =
241    $imap->buildflaglist();
242
243  $imap->append($mailbox,
244   $message, $fl);
245 }
```

### Listing 2: mbsetup

```
01 #!/usr/bin/perl
02 use strict;
03 use IMAP::Client;
04 use Sysadm::Install 0.23
05   qw(:all);
06
07 my $mailbox = "im_mailbox";
08
09 my $imap =
10   new IMAP::Client();
11 $imap->onfail('ABORT');
12 $imap->errorstyle('STACK');
13 $imap->debuglevel(0x01);
14
15 $imap->connect(
16  PeerAddr => 'localhost',
17  ConnectMethod => 'PLAIN'
18   )
19   or die "auth failure "
20   . $imap->error;
21
22 my $u  = getpwuid $>;
23 my $pw =
24   password_read("passwd: ");
25 $imap->authenticate($u, $pw);
26
27 $imap->onfail('ERROR');
28 $imap->delete($mailbox);
29 $imap->onfail('ABORT');
30
31 $imap->create($mailbox);
```

Most email clients support the IMAP protocol. If you want an easy to install IMAP server, I recommend Dovecot [4]. But whether you go for Cyrus, UW IMAP, or Dovecot, the *mbsetup* script (Listing 2) will create a new mailbox for chat email on your IMAP server.

If the debug level is set to *0x01*, as in *mbsetup*, *IMAP::Client* will additionally output the commands exchanged between the client and the server. This provides an excellent opportunity to study the quirky IMAP protocol, which assigns a unique number to each command and uses the same number for the response. This means that the server can start talking in between exchanges, for example, after an email lands in a mailbox that the client has signed up for. Thanks to the prepended number, the client can clearly distinguish between messages initiated by the server and messages responding to requests.

**Figure 4: The email client sees the completed messaging sessions on the IMAP server. The subject line in session five includes an *L* to indicate that a URL was exchanged.**

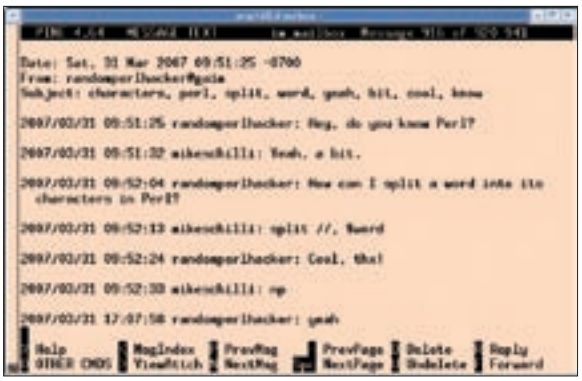

**Figure 5: The chat session text is available if the email client displays the mail text.**

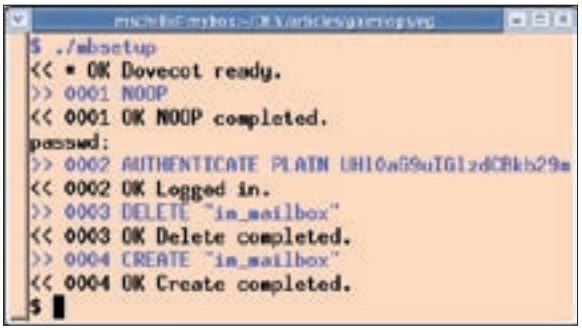


**Figure 6: The client and server communicate according to the IMAP protocol. Each request is assigned a unique numeric ID, which is sent again with the response.**

## Installation

If the IMAP server uses SSL to communicate (a must on the Internet and advisable on Intranets), you need to set the *ConnectMethod* parameter to *SSL*. *PLAIN* will work if the IMAP server has disabled SSL.

The CPAN modules used in *gaim2imap* require a couple of dependencies that a CPAN shell will automatically resolve. Line 166 of *gaim2imap* sets the languages to detect to English and Spanish (*en*, *es*). To reflect the languages you use, change the content of this anonymous hash.

## Time Zones

Gaim logs don't have the local time zone embedded, so it is up to the application parsing them to transform the time values to local time.

The *$tzone* variable in line 22 defines it, and if you don't happen to live in the time zone specified, you need to adapt it to your local setting.

When you start the *gaim2imap* daemon, you are prompted to enter the password the daemon can use to log on to the IMAP server with the user ID of the running process. By watching the logfile, you can check what the daemon is doing at any given time and how hard this automatic archivist is actually working.

If everything turns out as expected, the *im_mailbox* folder on the IMAP server should start to fill up with already-logged IM conversations after launching the program. While the daemon is active, it will pick up any chat sessions that take place. If the user is searching for information from a chat that happened the day before or many years ago, the email client provides convenient functions to search in the archive. Just think how easy it could be to find that elusive YouTube link a coworker told you about this morning. ■

### INFO

[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/82*

[2] Stopwords: *http://en.wikipedia.org/wiki/Stopword*

[3] Yahoo Term Extraction API: *http://developer.yahoo.com/search/content/V1/termExtraction.html*

[4] Dovecot, the secure IMAP server: *http://www.dovecot.org*