```
#!/usr/bin/perl -w

#
# ========== SOURCECODE FOR liisabotti IRC BOT ==========================
#
#       [ Copyright of this artwork is owner by Toni Willberg ]
#
# This file and it's contents is owned by Toni Willberg, toniw@iki.fi.
#
# This program is not open source, which means it's not yours, it's mine.
#
# I just wanted to give you an opportunity to see how this bot is made.
# You may learn, take ideas or routines from it, but you MAY NOT CLAIM
# THIS PROGRAM AS YOUR and also you HAVE TO give credits to me if you
# use something from this source.
#
# You don't have to pay anything to view this source. Atleast for now. :)
#
#
# Toni Willberg, toniw@iki.fi, http://www.iki.fi/toniw/
# Espoo, Finland, 2001-06-16
#
# ========================================================================
#









$| = 1;
use strict;


use Net::IRC;
use DB_File;
use Fcntl;
use Term::ANSIColor; # comes with perl 5.6.0


#
# ============ CONFIG ====================================================
#

my %config;
$config{debug} = 0; # print lot's of debug to STDOUT
$config{realnamecheck} = 0; # check sanity of joiners realname when he joins a c
hannel



$config{talkdelay} = 7; # how slow to write (seconds)
$config{sleepnow} = 1;  # sleep at beginning; wait before someone else talks

$config{idlelimit} = 360;       # after this many seconds bot changes
                                # channel if it's too empty
$config{peoplelimit} = 1;       # bot may leave channel if there are under n act
ive
                        # people left

$config{autoautomonologue} = 50;        # how often bot should talk by itself
                                # 0-1000, 0 = off, 999 = very often

$config{onleaveflamefreq} = 300;        # sometimes say something funny
                                        # about person who just left
```



```
$config{ontopicflamefreq} = 300;        # sometimes say something funny
                                        # about topic that was just changed




$config{autojoinfreq} = 2;      # how often bot should join randomly to
                                # known channel
                                # 0-1000, 0 = off, 999 = very often

$config{joinoninvite} = 1;      # join if someone invited

$config{autodialog} = 8;        # how often bot should start a dialog with
                                # someone
                                # 0-1000, 0 = off

$config{nokeyreply} = 100;      # how often bot should take part to
                                # conversation without known keyword

$config{autotopic} = 1;         # how often bot should change topic

$config{autoinvite} = 6;        # how often bot should invite people randomly



# COLOR configurations
$config{color_time} = 'green';
$config{color_normal} = 'white';
$config{color_channelname} = 'blue';
$config{color_nickhilight} = 'yellow';


#
# defaults for flood protection
#


$config{floodafter} = 6;  # This lets you specify the number of flooding
                          # messages you will see before on_flood is activated.

$config{floodrate} = 6;   # floodrate can be set to the number of
                          # messages per second you wish to  activate
                          # flooding. If messages from a user outpace
                          # floodrate for floodafter number of messages,
                          # on_flood is activated.  If floodrate is
                          # larger then floodafter, then you
                          # will end up seeing at least floodrate messages before
on_flood
                          # activation (If floodrate is 5 and floodafter is 3 then
 you must
                          # receive at least 5 messages before the on_flood rate c
an be 5
                          # messages per second).

$config{floodrepeat} = 3; # How many equal lines may occur on channel before
                          # on_flood is activated.




# which servers to use
my @servers = (
                'irc.stealth.net',
                'irc.kolumbus.fi',
                'irc.kolumbus.fi',
                'irc.cs.hut.fi',
```

```
                'irc.eunet.fi',
                'irc.funet.fi',
                'irc.sgic.fi'
                );

# which nicks to use (if specified is not available)
my @nicks = (
              'liisa',
              'aliisa',
              'lissukka',
              'elisa',
              'eliisa'
              );



# who can command us
my @masters = (
                [redacted]                                    'nick1!^username1@hostname',
                                                              'nick2!^username2@hostname',
                [redacted],
              );

my @dontmsg = (
                [redacted]          'friend1', 'friend2',
                [redacted]
                [redacted]
                [redacted]
                [redacted]
                );



#
# ================================================================
#


###############
#
# Last check in     : $Date: 2001/06/19 18:56:42 $
# Revision number   : $Revision: 2.17 $
# State             : $State: Exp $
#
# $Log: liisabotti,v $
# Revision 2.17  2001/06/19 18:56:42  twillber
# Now saying something after joined to some channel on invite.
#
# Revision 2.16  2001/06/18 23:08:43  twillber
# Changed phrases -> arrays
#
# Revision 2.15  2001/06/18 22:52:25  twillber
# Moved all tie -routines to 4 functions. Removed about 200 lines of code :D
#
# Revision 2.14  2001/06/17 22:32:24  twillber
# Now saying funny things about people who just got kicked out from channel.
#
# Revision 2.13  2001/06/17 19:24:39  twillber
# Fixed filehandle handling code. Now a bit more secure.
#
# Revision 2.12  2001/06/17 18:49:20  twillber
# Now flaming person who just left channel/irc.
#
```



```
# Revision 2.11  2001/06/17 18:13:24  twillber
# Fixed leaveChannel -debugging.
#
# Revision 2.10  2001/06/17 18:01:02  twillber
# Added some kind of support for CTCP ACTION.
#
# Revision 2.9  2001/06/15 21:32:56  twillber
# Added copyright notice before publishing the source.
#
# Revision 2.8  2001/06/13 19:55:11  twillber
# Changed logfile name.
#
# Revision 2.7  2001/06/13 19:18:09  twillber
# - Moved console log from STDOUT to different file.
# - Fixed /invite -code.
#
# Revision 2.6  2001/06/06 22:06:56  twillber
# added untie for phrases_questions ;)
#
# Revision 2.5  2001/06/06 21:36:21  twillber
# added new phrasetype: phrases_channels
#
# Revision 2.4  2001/06/06 20:54:25  twillber
# Fixed autojoinfreq.
# Now does WHOIS when someone joins.
# Stores all channels of nicks we whoised.
# etc...
#
# Revision 2.3  2001/06/06 19:32:47  twillber
# Fixed on_quit -bug. Now recognizes all quits.
#
# Revision 2.2  2001/06/05 18:37:48  twillber
# New version.
#
# Revision 2.1  2001/06/05 18:35:37  twillber
# Initial revision
#
# Revision 1.55  2001/03/12 18:35:26  twillber
# Minor config
#
# Revision 1.54  2001/03/11 15:25:18  twillber
# Minor config.
#
# Revision 1.53  2001/03/11 00:40:05  twillber
# Removed all STDOUT -prints and replaced with &log();
#
# Revision 1.52  2001/03/10 23:12:45  twillber
# Now using configs more wisely.
# Added colors to console log.
#
# Revision 1.51  2001/01/23 23:59:56  twillber
# added conf -command
# fixed places
#
# Revision 1.49  2000/11/04 09:07:08  twillber
# Changed way of changing topic randomly
#
# Revision 1.48  2000/10/31 19:50:21  twillber
# Added option "autojoin"
#
# Revision 1.47  2000/09/10 16:55:33  twillber
# Added couple of new phrases
#
# Revision 1.46  2000/09/10 16:12:48  twillber
# Someone(tm) forgot to commit last changes...
#
# Revision 1.45  2000/08/03 21:39:06  twillber
# Fixed "minor" problems with some stupid things...
#
```

```
# Revision 1.44  2000/07/30 16:36:53  twillber
# Added watch.
#
# Revision 1.43  2000/07/30 15:59:53  twillber
# Added jokes.
#
# Revision 1.42  2000/05/24 21:12:38  twillber
# Fixed bugs.
#
# Added unban etc...
#
# Revision 1.41  2000/05/20 22:52:57  twillber
# Fixed some things.
# Checks if the bot is an op before even trying to do a op's job.
# etc...
#
# Revision 1.40  2000/05/18 22:59:49  twillber
# Understands adjectives and numbers.
#
# Revision 1.39  2000/05/18 21:49:43  twillber
# Fixed dead pidfiles.
# Added automatic nick change on nick reserved -case.
# Added REBOOT -command.
#
# Revision 1.38  2000/05/16 21:02:14  twillber
# Added pid-file,
# improved some functions etc.
#
# Revision 1.37  2000/05/13 01:48:15  twillber
# Major changes.
# More intelligent joining and leaving.
# Handles mode changes correctly.
# etc...
#
# Revision 1.36  2000/05/12 21:58:45  twillber
# some changes
#
# Revision 1.35  2000/05/07 20:22:37  twillber
# Learns more phrases.
# Fixed filehandling.
# etc...
#
# Revision 1.34  2000/05/07 16:28:38  twillber
# Fixed bug in nicklist and channellist handling.
#
# Revision 1.33  2000/05/07 15:19:37  twillber
# On_quit -event works now.
# Minor fixes.
#
# Revision 1.32  2000/05/05 17:45:53  twillber
# Fixed flood and ops-kick
#
# Revision 1.31  2000/05/03 18:32:43  twillber
# Random joining works again.
#
# Revision 1.30  2000/05/02 22:39:51  twillber
# Fixed several routines.
# Now randomChannel, randomNick etc. work
#
# Revision 1.29  2000/04/30 22:59:21  twillber
# Improved timer. Now allows multiple timers simultaneously without blocking.
#
# Revision 1.28  2000/04/19 23:18:43  twillber
# Improved flood and garbage inspection
#
# Revision 1.27  2000/04/05 08:47:06  twillber
# Bugfixes for last night ;)
#
# Revision 1.26  2000/04/05 00:59:34  twillber
```



```
# MAJOR rearranging of code;
# - changed way of logging
# - supports multiple simultaneous channels
# - disabled some features (auto-join-on-bored) for now
# - bugfixes
#
# Revision 1.25  2000/04/04 19:26:36  twillber
# Advanced parser a bit
#
# Revision 1.24  2000/04/04 17:57:16  twillber
# Better floodprotector
#
# Revision 1.23  2000/04/03 22:09:53  twillber
# Disabled Away for now....
# Disable auto-messaging
#
# Revision 1.22  2000/04/03 21:39:08  twillber
# Automaticly sets Away messages.
# Better question handling.
#
# Revision 1.21  2000/04/03 19:12:51  twillber
# now changes topics...
#
# Revision 1.20  2000/04/02 14:57:04  twillber
# Bugfix to previous.
# Changed frequency of random messages.
#
# Revision 1.19  2000/04/02 14:50:25  twillber
# Now learns all questions and curses...
#
# Revision 1.18  2000/04/01 21:02:38  twillber
# Added more phrases and repeation detection.
#
# Revision 1.17  2000/03/31 16:21:38  twillber
# Added some handlers for errors.
#
# Revision 1.16  2000/03/31 11:07:58  twillber
# minor bugfixes
#
# Revision 1.15  2000/03/30 19:43:25  twillber
# Separated log files for messages and channels.
# Using same log file per channel/user per day.
#
# Revision 1.14  2000/03/26 22:52:24  twillber
# Automaticly messages people...
#
# Revision 1.13  2000/03/22 15:34:15  twillber
# DONTGOCHANNEL -list
#
# Revision 1.12  2000/03/21 07:04:21  twillber
# Now gathers names on channel only when joining and when someone joins.
#
# Revision 1.11  2000/03/21 06:39:43  twillber
# Minor fixes.
#
# Revision 1.10  2000/03/20 19:24:12  twillber
# Added leave-commans etc.
#
# Revision 1.9  2000/03/20 17:03:12  twillber
# Fixed bugs. ;)
#
# Revision 1.8  2000/03/19 22:00:22  twillber
# Works ok.
#
# Revision 1.7  2000/03/19 19:50:22  twillber
# Better behaviour, does not talk so much if not talked to.
#
# Revision 1.6  2000/03/19 19:08:24  twillber
# Changed lot's of things.
```

```
#
# Revision 1.5  2000/03/19 17:00:01  twillber
# Now handles private conversations.
#
# Revision 1.4  2000/03/19 15:56:49  twillber
# Works nice at this point.
# Moves around channels if bored or kicked.
#
# Revision 1.3  2000/03/19 02:03:29  twillber
# Lots of changes ;)
#
# Revision 1.2  2000/03/18 20:58:44  twillber
# -
#
# Revision 1.1  2000/03/18 19:27:15  twillber
# Initial revision
#
#
###############

#
# OPEN CONSOLE LOG
my ($sec,$min,$hour,$dd, $mm, $yyyy) = localtime(time());
$mm++;
$yyyy += 1900;
$mm = "0$mm" if ($mm < 10);
$dd = "0$dd" if ($mm < 10);

my $consolelogfile = "logs/console/current.log";
open (CONSOLELOG, ">>$consolelogfile") || die ("*** Cannot open consolelog
'$consolelogfile'. Quitting.\n\n");

select CONSOLELOG; $| = 1; # unbuffer

print CONSOLELOG "#pid $$ started...\n";
#
##


# on signal, quit (1)
use sigtrap 'handler', \&quitter, 'normal-signals';


# check out pid file

# this is used as a pid file
my $pidfile = "liisabotti.pid";


if (-e $pidfile) {
  my $opid;

  &log("Pid-file found. Checking content...");
  open (PF, "$pidfile") || die ("Can't open PIDFILE $pidfile: $!\n");

  $opid = <PF>;

  if ($opid ne $$) {
    &log("PID found from pid-file does not match, checking if pid $opid is running...");

    my @pidlist = `ps $opid`;

    if ($pidlist[1]) {
      &log("The bot is running with pid $opid. We will not start.");
      exit;
    }
    else {
      &log("Process $opid has died. Removing file and starting new bot.");
      open (PF, ">$pidfile") || die ("Can't open PIDFILE $pidfile: $!\n");
```



```
      print PF "$$";
      close PF;
    }

  }
  else {
    &log("Pid matched, the bot is dead, overwriting");

    open (PF, ">$pidfile") || die ("Can't open PIDFILE $pidfile: $!\n");
    print PF "$$";
    close PF;
  }

}
else {
  &log("Pid-file not found. Creating new...");

  open (PF, ">$pidfile") || die ("Can't open PIDFILE $pidfile: $!\n");
  print PF "$$";
  close PF;

}



my $irc; # global object for irc sockets etc.
my $KEEPRUNNING;


#
#####################################################################
#

my $channelwonder = 0;  # pointer to a channel bot should join next
my $nicklistsize = 10; # how many nicks to remember
my $publicbuffersize = 8; # How many lines per channel bot must keep in
                          # buffer. This effects also to flood protection,
                          # so try to keep this at least over 10.




my %msghash;
my %inmsghash;


my %publicbuffer;


my $time_msg = time();
my $time_start = time(); # when the bot was started


my $lastmsg = time();
my $botlastmsg = (); # bot's last line

my $time_say = time();  # last time bot said something
my $lasttalked = time();


# where the phrase files are
my $arrayspath = "arrays/";

# where the word files are
my $wordspath = "words/";
```

```
# some private information
my $version = 'Liisabotti. http://liisa.on.peelo.com/';
my $finger = 'Liisabotti. http://liisa.on.peelo.com/';
my $userinfo = 'Liisabotti. http://liisa.on.peelo.com/';


# take command line args
my $bot_server = $ARGV[0];
my $bot_nick = $ARGV[1];
my $bot_chan = $ARGV[2];
my $bot_realname = $ARGV[3];
my $bot_username = $ARGV[4];
my $AUTOJOIN = $ARGV[5]; # remove later!
$bot_chan = lc($bot_chan);
$bot_nick = lc($bot_nick);



my $AUTOTOPICCHANGE = 0; # if bot should change topic of channels randomly


my $conn = "";

# active users on channels that bot is in
my %channelnicks = ();

# who was last to talk on channel?
my %chantalk = ();


# where to log
my $PUBLOGPATH = "logs/channels/";
my $MSGLOGPATH = "logs/messages/";
my $LOGFILE = "";



my $time_public = time();       # last time someone said something

# the channels that bot is an op
my %opchannels = ();

# the channels that bot is in
my %onchannels = ();



# all phrases all stored in this hash
my %arrays = ();


#
# tie some files
#



#
# TIE all arrays
#

# LIST of arrayfiles to be tied
my @tieablearrays = (
                      'ignores',
                      'known_channels',
                      'dontgochannels',
                      'phrases_answers',
                      'phrases_onleaves',
```



```
                      'phrases_ontopics',
                      'phrases_unknown',
                      'phrases_channels',
                      'phrases_deny',
                      'phrases_goodnight',
                      'phrases_greet',
                      'phrases_smileys',
                      'phrases_bored',
                      'phrases_ignorance',
                      'phrases_wonders',
                      'phrases_agrees',
                      'phrases_mymusic',
                      'phrases_whoami',
                      'phrases_myage',
                      'phrases_curse',
                      'phrases_nosex',
                      'phrases_whatis',
                      'phrases_quiet',
                      'phrases_questions',
                      'phrases_joke',
                      'phrases_oninvite',
                      'words_adjectives',
                      'words_numbers'
                    );

# tie all given arrays
sub tieAllArrays {
  foreach my $cat (@tieablearrays) {
    &log("Trying to tie array '$cat'...");
    $arrays{"$cat"} = [];
    &tieArray($arrays{"$cat"}, "$cat");
  }
}

# untie all listed arrays
sub untieAllArrays {
  foreach my $cat (@tieablearrays) {
    &log("Trying to tie array '$cat'...");
    $arrays{"$cat"} = [];
    &untieArray($arrays{"$cat"}, "$cat");
  }
}


# go
&tieAllArrays();



#
#
#

my $lastline = "";



if ( !$bot_server || !$bot_nick || !$bot_chan || !$bot_realname) {
  die ("
Usage: $0 server nick channel realname username autojoin
Channel without #!

Ex: $0 irc.kolumbus.fi lissu peelo \"Liisa Botti\" lissukka 1

Autojoin: 1=Bot may randomly join channels, 0=Not.
");
}
```

```
my @nicklist; # nicks at channel
my %stats = ();




#
#
#
#
#
#


#
# write to consolelog
# %STATUS: OK
sub log {
  my ($msg) = @_;
  my ($sec, $min, $hour, $day, $month, $year) = &getTime( localtime(time() ) );

  my $color_time = color($config{color_time});
  my $color_normal = color($config{color_normal});
  my $color_channelname = color($config{color_channelname});
  my $color_nickhilight = color($config{color_nickhilight});

#  my $clog = ($color_time."[$hour:$min:$sec]".$color_channelname ." {CONSOLE} "
.$color_normal."$msg");
  my $clog = ("[$hour:$min:$sec] {CONSOLE} $msg");

  print CONSOLELOG "$clog\n";
}




#
# write something to log file
# %STATUS: OK
sub logChannel {
  my ($channel, $msg) = @_;

  if (!$channel || !$msg) {
    &log("*** Not enough parameters for &logChannel ($channel, $msg)");
    return;
  }


  # this is necessary for using scalar as filehandle!!!
  no strict "refs";

  $channel =~ s/\#//g; # remove # from channel
  $channel =~ s/\///g; # remove / from channel
  $channel = lc($channel);


  my ($sec, $min, $hour, $day, $month, $year) = &getTime( localtime(time() ) );

  my $color_time = color($config{color_time});
  my $color_normal = color($config{color_normal});
  my $color_channelname = color($config{color_channelname});
  my $color_nickhilight = color($config{color_nickhilight});


  # file log
#  my $flog = ($color_time."[$hour:$min:$sec]".$color_channelname ." {#$channel}
```



```
 ".$color_normal."$msg");
  # no color
  my $flog = ("[$hour:$min:$sec]"." {#$channel} "."$msg");


  # hilight nick
  if ($msg =~ /$bot_nick/i) {
    $msg = $color_nickhilight.$msg.$color_normal;
  }

  # console log
  my $clog = ($color_time."[$hour:$min:$sec]".$color_channelname ." {#$channel} ".$color
_normal."$msg");

  # print on monitor
#  print CONSOLELOG "$clog\n";
  print CONSOLELOG "$flog\n";

  # write to channel filehandle
#  my $chfh = "logfile-$channel";
  my $chfh = &getFileHandle($channel);
  print $chfh "$flog\n";
}




#
# append messages from same nick to same logfile
#
sub logMessage {
  my ($nick, $msg) = @_;


  my ($sec, $min, $hour, $day, $month, $year) = &getTime( localtime(time()) );

  my $fnick = lc($nick);

  if ($fnick =~ /^\./) {
    &log("*** ILLEGAL NICK '$fnick'");
    return;
  }

  my $mgfile = "$fnick.$year\_$month\_$day.txt";

  open (MLOG, ">>$MSGLOGPATH/$mgfile") || die ("Can't open MLOG: $mgfile: $!");

  my $ll = "[$hour:$min:$sec] {$nick} $msg\n";

  # print on monitor
  print CONSOLELOG "$ll";

  # write to file
  print MLOG $ll;

  close MLOG || &log("Can't close MLOG: $mgfile: $!");
}



#
# TIE a array
#
sub tieArray {
  my ($ptr, $name) = @_;
  tie @$ptr, "DB_File", "$arrayspath/$name", O_RDWR|O_CREAT, 0640, $DB_RECNO || die "
Cannot open '".$name."': $!\n" ;
  return $ptr;
}
```

```
#
# UNTIE a array
#
sub untieArray {
  my ($ptr) = @_;
  untie @$ptr;
}

#
# kick user from channel
#
sub kickUser {
  my ($self, $channel, $nick, $reason) = @_;

  if (!$self || !$channel || !$nick || !$reason) {
    &log("*** ERROR: not enough parameters for &kickUser()");
  }

  if ($opchannels{$channel}) {
    $self->kick("\#$channel", $nick, $reason);
  }
  else {
    &log("* Not an op on $channel! Can't do kick \#$channel $nick: $reason");
  }

}

#
# ignore user
#

sub ignoreUser {
  my ($nick) = @_;

#  push @ignores, $1;
  my $ptr = $arrays{'ignores'};
  push @$ptr, $1;
}



#
# ban user on channel (for $length seconds [optional])
#
sub banUser {
  my ($self, $channel, $nick, $length) = @_;

  if (!$self || !$channel || !$nick) {
    &log("*** ERROR: not enough parameters for &banUser()");
  }

  if ($opchannels{$channel}) {
    $self->mode("\#$channel", "+b", $nick);
  }
  else {
    &log("* Not an op on $channel! Can't do ban \#$channel $nick");
  }


  $length = 60 if (!$length);

  # if length was give, create timer to unban the user
  $self->parent->queue(time + $length,
                       sub{
                         &unbanUser($self, $channel, $nick);
                       }, $self);
```



```
}

#
# unban user on channel
#
sub unbanUser {
  my ($self, $channel, $nick) = @_;

  if (!$self || !$channel || !$nick) {
    &log("*** ERROR: not enough parameters for &banUser()");
  }

  if ($opchannels{$channel}) {
    $self->mode("\#$channel", "-b", $nick);
  }
  else {
    &log("* not an op on $channel! Can't unban $nick");
  }

}




#
# deop user from channel
#
sub deopUser {
  my ($self, $channel, $nick) = @_;

  if (!$self || !$channel || !$nick) {
    &log("*** ERROR: not enough parameters for &deopUser()");
  }

  if ($opchannels{$channel}) {
    $self->mode("\#$channel", "-o", $nick);
  }
  else {
    &log("* not an op on $channel! Can't deop $nick");
  }

}

#
# op user on channel
#
sub opUser {
  my ($self, $channel, $nick) = @_;

  if (!$self || !$channel || !$nick) {
    &log("*** ERROR: not enough parameters for &opUser()");
  }

  if ($opchannels{$channel}) {
    $self->mode("\#$channel", "+o", $nick);
  }
  else {
    &log("* not an op on $channel! Can't op $nick");
  }

}



#
# invite user to channel
#
```

```
sub inviteUser {
  my ($self, $channel, $nick) = @_;

  if (!$self || !$channel || !$nick) {
    &log("*** ERROR: not enough parameters for &inviteUser()");
  }

  if (!$onchannels{$channel}) {
    &log("*** Duh! I'm not on \#$channel. Can't invite anyone there!");
    return;
  }

  &logChannel($channel, "*** Going to invite $nick to \#$channel");
  $self->invite($nick, "\#$channel");
}



#
# add $nick to nicklist of $channel
#
sub addNick {
  my ($channel, $nick) = @_;

  if (!$channel || !$nick) {
    &log("*** Not enough parameters on &addNick($channel, $nick)");
  }

  $nick = lc($nick);


  if ($nick =~ /bot|away/i) {
    # dont add people who are away or bots
    return;
  }

  my $nbn = lc($bot_nick);

  if ($nick eq $nbn) {
    &log("&addNick: not adding own nick") if ($config{debug} > 3);
    return;
  }

  &log("Adding '$nick' to \#$channel list") if ($config{debug} > 3);


  # check that nick is not on the list already
  my $nck;
  foreach $nck (@{$channelnicks{$channel}}){
    if ($nick eq $nck) {
      # nick was already on list, don't add again
      &log("$nick was already on $channel nicklist") if ($config{debug} > 3);
      return;
    }
  }

  # add nick to channel list
  $nick = lc($nick);
  push @{$channelnicks{$channel}}, $nick;
  &log("Added '$nick' to '\#$channel' nicklist. Now ". $#{$channelnicks{$channel}} ." people on ch
annel") if ($config{debug});

}



#
# remove $nick from nicklist of $channel
#
sub removeNick {
```



```
  my ($channel, $nick) = @_;

  # we user lower case nicks internally
  $nick = lc($nick);

  &log("Removing '$nick' from \#$channel list") if ($config{debug});


  my ($nc, @newchn);

  for ($nc=0; $nc <= $#{$channelnicks{$channel}}; $nc++) {
    if (@{$channelnicks{$channel}}[$nc] eq $nick) {
      splice @{$channelnicks{$channel}}, $nc, 1; # remove this nick
      my $pcount = $#{$channelnicks{$channel}};
      &log("Removed '$nick' from '\#$channel'. $pcount people still on channel.");
      &log("list is now: '@{$channelnicks{$channel}}'") if ($config{debug});

      if ($pcount < $config{peoplelimit} && ($channel ne $bot_chan) ) {
        &log ("Leaving channel '\#$channel' because it has under $config{peoplelimit} people left");
        &leaveChannel($conn, $channel);
      }

      last; # no need to continue
    }
  }


}



#
# return random nick from specified channel
#
sub randomNick {
  my ($channel) = shift;;

  if (!$channel) {
    &log("Not enough parameters for &randomNick($channel)") if ($config{debug});
    return
  }

  my $max = $#{$channelnicks{$channel}};


  my $nr = int(rand($max+1));
  my $nick = @{$channelnicks{$channel}}[$nr];
  $nick = lc($nick);


  &log ("randomnick: '\#$channel', nick: '$nick'") if ($config{debug});
  &log("list: '@{$channelnicks{$channel}}'") if ($config{debug});
  return $nick;

}



#
# return random nick from our %onchannels
#
sub randomChannel {

  my $channel = "";
  my ($chc, $count) = 0;
  my $chs = scalar keys %onchannels;
  my $retch = int(rand($chs));

  foreach $channel ( keys %onchannels) {
```

```
    if ($chc == $retch) {
      &log("randomchannel ($retch) = '$channel'") if ($config{debug} > 2);
      return $channel;
    }
    $chc++;
  }

  # return last; for some reason we did not get any channel - this is not
  # supposed to happen, only on when bot is not on any channel
  return $channel;
}




#
# return random channel from known channels
#
sub randomKnownChannel {

  my $channel;

  my $pptr = $arrays{'known_channels'};
  my $max = $#$pptr;
  my $chnr = int(rand($max)+1);
  $channel = @$pptr[$chnr];

  &log("*** returning randomKnownChannel: $channel") if ($config{debug});

  return $channel;
}


#
# return next known channel
#
sub nextKnownChannel {

  my $channel;

  $channelwonder++; # add pointer

  my $pptr = $arrays{'known_channels'};

  # check if needed to loop over
  if (!@$pptr[$channelwonder]) {
    $channelwonder = 0;
  }

  $channel = @$pptr[$channelwonder];

  &log("nextKnownChannel: $channel") if ($config{debug});

  return $channel;
}




#
# return randon phrase from given \@list
#
sub randomPhrase {
  my ($nick, $list) = @_;

  my $max = $#$list + 1;
  my ($nr, $msg) = "";
```



```
  if (!$list) {
    &log("NO LIST '$list' &randomPhrase");
    return;
  }

  $nr = int(rand($max)+1);
  $msg = @$list[$nr];

  if (!$msg || $msg eq '') {
    # this is ok
    return;
  }

  # local

  if ($msg =~ /\%randsomething/) {
#     my $nmsg = @phrases_unknown[int(rand($#phrases_unknown)+1)];
    my $pptr = $arrays{'phrases_unknown'};
    my $nmsg = @$pptr[int(rand($#$pptr)+1)];
    $msg =~ s/\%randsomething/$nmsg/ge;
  }

  if ($nick) {
    # to $nick
    $msg =~ s/\%nick/$nick/g;

  } # if

  # convert any leftover stuff...
  $msg =~ s/\%nick,|\%nick:|\%nick\ |\%nick//g; # get rid of %nick: and %nick,

  if (!$msg) {
    &log("INTERNAL ERROR: NULL MESSAGE &randomPhrase");
    return;
  }

  return $msg; # return the message


}




#
# says phrase from given list
# input: $self, @list
#
sub sayPhrase {
  my ($self, $channel, $nick, $list) = @_;
  my $max = $#$list+1;
  my ($nr, $msg) = "";

  if (!$list) {
    &log("NO LIST '$list' &sayPhrase");
    return;
  }

  # get random message from given list
  $nr = int(rand($max));
  $msg = @$list[$nr];

  &log("MSG($nick): '$msg'") if ($config{debug});

  # null message, dont say it
  if (!$msg || $msg eq '') {
    &log("*** No message on &sayPhrase()");
    return;
  }
```

```
  # convert %randsomething -tag to something
  if ($msg =~ /\%randsomething/) {
    my $pptr = $arrays{'known_unknown'};

    my $nmsg = @$pptr[int(rand($#$pptr)+1)];
    $msg =~ s/\%randsomething/$nmsg/ge;
    &log("*** Replaced \%randsomething tag with '$nmsg' at &sayPhrase()");
  }

  # convert %adjectives
  if ($msg =~ /\%adjective/) {
    my $pptr = $arrays{'words_adjectives'};
    my $nmsg = @$pptr[int(rand($#$pptr)+1)];
    $msg =~ s/\%adjective/$nmsg/ge;
    &log("*** Replaced \%adjective tag with word '$nmsg' at &sayPhrase()");
  }

  # convert %number
  if ($msg =~ /\%number/) {
    my $pptr = $arrays{'words_numbers'};
    my $nmsg = @$pptr[int(rand($#$pptr)+1)];
    $msg =~ s/\%number/$nmsg/ge;
    &log("*** Replaced \%number tag with word '$nmsg' at &sayPhrase()");
  }



  if ($nick) {
    # to $nick

    # insert %nick in front of message if it is missing for some reason...
    if ($msg !~ /\%nick/g) {
      $msg = "\%nick: $msg";
    }

    $msg =~ s/\%nick/$nick/g;

  } # if
  else {
    # to channel perhaps? :)
    $msg =~ s/\%nick,|\%nick:|\%nick\ //g; # get rid of %nick: and %nick,

  }

  if (!$msg) {
    &log("INTERNAL ERROR: NULL MESSAGE &sayPhrase");
    return;
  }

  &say($self, $channel, $msg); # say it you lame bot ;)

}
###



#
# say something on channel
#
sub say {
  my ($self, $channel, $msg) = @_;

  $msg = substr ($msg, 0, 200); # cut from 200 chars to avoid excess flood :)
  $channel = lc($channel);

  if (!$msg) {
    &log("*** NULL MESSAGE &say()");
    return;
```



```
  }

  if (!$onchannels{$channel}) {
    &log("*** '\#$channel' - I'm not on that channel!");
    return;
  }

  # we talked last on this channel
  $chantalk{$channel} = $bot_nick;

  # check that bot does not repeat
  if ($lastline eq $msg) {
    &log("{msg loop \"$msg\", skipping}") if ($config{debug});
  }
  else {
    $time_say = time();

    $stats{$bot_nick}++;

    # convert random something
    if ($msg =~ /\%randsomething/gi) {
      $msg =~ s/\%randsomething/&randomPhrase(undef, $arrays{'phrases_unknown'})
/ge;
    }

    # convert random nicks
    if ($msg =~ /\%randnick/gi) {
      $msg =~ s/\%randnick/&randomNick($channel)/ge;
    }

    # convert random nicks
    if ($msg =~ /\%randomchannel/gi) {
      my $rknchan = &randomKnownChannel();
      $rknchan = "#".$rknchan;
      $msg =~ s/\%randomchannel/$rknchan/ge;
    }

    # convert adjectives
    if ($msg =~ /\%adjective/gi) {
      $msg =~ s/\%adjective/&randomPhrase(undef, $arrays{'words_adjectives'})/ge
;
    }

    $msg =~ s/\%bot_nick/$bot_nick/g;
    $msg =~ s/\%nick|%randnick//g;

    $msg =~ s/\ ,//g;
    $msg =~ s/^,\ //g;
    $msg =~ s/^,//g;
    $msg =~ s/\ \ |\ \ \ /\ /g;
    $msg =~ s/^\ //g;
    $msg =~ s/\ a\,//g;
    $msg =~ s/\ a\://g;

    # correct some errors on phrases
    $msg =~ s/\ lla\ /"\ ".&randomNick($channel)."\:lla "/ge;
    $msg =~ s/\ lle\ /"\ ".&randomNick($channel)."\:lle "/ge;
    $msg =~ s/\ n\ /"\ ".&randomNick($channel)."\:n "/ge;
    $msg =~ s/\ a\ /"\ ".&randomNick($channel)."\:a "/ge;

    # what exactly is this? :)
    if ( $config{automonologue} && &ifrand($config{automonologue}, 'automonologue')
) {
      $msg = "$msg " .&randomPhrase(undef, $arrays{'phrases_smileys'});
    }

    # this code put's 'sub' code into queue, and it will be executed later
    # by Net::IRC
```

```
    $self->parent->queue(time + $config{talkdelay},
                         sub{
                           $self->privmsg("#$channel", "$msg");
                           &logChannel($channel, "[SAY] '$msg'");
                           $lasttalked = $bot_nick;
                         }, $self);


  }

}




#
# messages phrase from given list
# input: $self, @list
#
sub msgPhrase {
  my ($self, $nick, $list) = @_;
  my $max = $#$list + 1;
  my ($nr, $msg);

  if (!$list) {
    &log("*** NO LIST &msgPhrase");
    return;
  }

  if (!$nick) {
    &log("*** NO NICK &msgPhrase");
    return;
  }


  # check that bot is not messaging to forbidden ones
  foreach my $ignos (@dontmsg) {
    if ($nick eq $ignos) {
      &log("*** $nick is dontmsg. I can't talk to him too.");
      return;
    }
  }

  $nr = int(rand($max)+1);
  $msg = @$list[$nr];

  if (!$msg) {
    &log("*** NULL MSG!!!");

    $nr = int(rand($max)+1);
    $msg = @$list[$nr];

    if (!$msg) {
      &log("*** STILL NULL MSG!!!");
      return;
    }

  }

  # local
  $msg =~ s/\%nick\:|\%nick\,|\%nick//g;
  $msg =~ s/\%botchan/$bot_chan/g;
  $msg =~ s/\%randnick//g;

  $msg =~ s/\ \ /\ /g; # remove double spaces
  $msg =~ s/^\ //g; # remove leading spaces
  $msg =~ s/^\ //g;
```



```
  $msg =~ s/^,\ //g;
  $msg =~ s/\ ,\ //g;

  # convert random something
  if ($msg =~ /\%randsomething/) {
    $msg =~ s/\%randsomething/&randomPhrase(undef, $arrays{'phrases_unknown'})/g
e;
  }

  # convert adjectives
  if ($msg =~ /\%adjective/g) {
    $msg =~ s/\%adjective/&randomPhrase($arrays{'words_adjectives'})/ge;
  }



  $self->parent->queue(time + $config{talkdelay},
                       sub{
                         $self->privmsg("$nick", "$msg");
                         &logMessage($nick, "[MSG > $nick] $msg");
                       }, $self);

}


#
# inserts new line to given phrase list
#
sub learnPhrase {
  my ($line, $list) = @_;

  if (!$line) {
    &log("learnPhrase needs a line! ");
  }

  $line =~ s/$bot_nick/\%nick/gi;
  $line =~ s/^(.*?)\://gi;
  $line =~ s/^(.*?)\,//gi;
  $line =~ s/^\ //gi;

  &log("\tLEARN: '$line'") if ($config{debug});

  push @$list, $line;
}


#
# change topic
#
sub changeTopic {
  my ($self, $channel, $topic) = @_;

  if ($opchannels{$channel}) {
    $self->topic("\#$channel", $topic);
  }
  else {
    &log("* not an op on $channel! Can't change topic to '$topic'");
  }


}


#
# remembers all #channels for later use... ;)
#
sub learnChannel {
  my ($line, $list) = @_;
```

```
  &log("*** learnChannel($line)") if ($config{debug});


  my $newchan = "";

  if ($line =~ /\#(.*)/) {
    # if found #channel
    $newchan = $1;

    $newchan =~ /(.*?)\ /;
    $newchan = $1;

    $newchan =~ /(.*?)\:/;
    $newchan = $1;

    $newchan =~ /(.*?)\-/;
    $newchan = $1;

    # illegal characters in channel
# tähän vähän tarkistusta!!!
#    if ($newchan =~ /[a-z.-_]/) {
#      &log("*** Illegal char in chan name: $newchan");
#      return;
#    }


    # check if channel was forbidden
    my $chn = "";
    foreach $chn ($arrays{'dontgochannels'}) {
      if ($newchan eq $chn) {
        &log("DONTGOCHANNEL: '$newchan'") if ($config{debug});
        return;
      }
    }

    # check if channel was already in list
    $chn = "";
    foreach $chn (@$list) {
      if ($newchan eq $chn) {
        &log("WAS ALREADY IN LIST: '$newchan'") if ($config{debug});
        return;
      }
    }

    &log("LEARNT NEW CHANNEL: '$newchan'") if ($config{debug});
    push @$list, $newchan;


  }
  else {
    # can't parse, too bad...
    &log("Can't find channel from string '$line'.");
  }

}



#
# message something to someone
#
sub msg {
  my ($self, $to, $msg) = @_;

  if (!$to) {
    &log ("*** NO NICK &msgPhrase");
    return;
  }
```



```
  # check that bot is not messaging to forbidden ones
  foreach my $ignos (@dontmsg) {
    if ($to eq $ignos) {
      &log ("*** $to is dontmsg. I can't talk to him too.");
      return;
    }
  }

  # check that bot does not repeat
  if ($botlastmsg && $botlastmsg eq $msg) {
    &log("{msg loop, ignoring this line}") if ($config{debug});

  }
  else {


    $msg =~ s/\%randnick//ge;


    if ($msg =~ /\%randsomething/) {
      my $pptr = $arrays{'phrases_unknown'};
      my $nmsg = @$pptr[int(rand($#$pptr)+1)];
      $msg =~ s/\%randsomething/$nmsg/ge;
    }

    # get bot channel
    if ($msg =~ /\%botchan/g) {
      $msg =~ s/\%botchan/\#$bot_chan/g;
    }

    # get bot nick
    if ($msg =~ /\%botnick/g) {
      $msg =~ s/\%botnick/\$bot_nick/g;
    }

    # get bot nick
    if ($msg =~ /\%nick/g) {
      $msg =~ s/\%nick/\$to/g;
    }

    $msg =~ s/\ ,//g;
    $msg =~ s/^\ //g;
    $msg =~ s/^,\ //g;
    $msg =~ s/^,//g;

    $botlastmsg = $msg; # save last line;


    $self->parent->queue(time + $config{talkdelay},
                         sub{
                           $self->privmsg("$to", "$msg");
                           &logMessage($to, "[MSG > $to] $msg");
                         }, $self);

  }

}


# What to do when the bot successfully connects.
sub on_connect {
  my $self = shift;

  &joinChannel($self, $bot_chan);

}
```

```
# just before quit
sub do_quit {

  my $channel = ();
  foreach $channel (keys %onchannels) {

    $config{talkdelay} = 0;
    &leaveChannel($conn, $channel, 'Hei hei kullannuput, mä palaan vielä ;)');

  }


  # disconnect from server
  $irc->removeconn($conn);
  sleep 10;
}



sub leaveChannel {
  my ($self, $channel, $msg) = @_;


  # this is necessary for using scalar as filehandle!!!
  no strict "refs";

  $channel = lc($channel);
  $channel =~  s/\#//;

  if (!$self || !$channel) {
    &log("*** ERROR: not enough parameters for &leaveChannel()");
  }


  $msg = "" if (!$msg);

  if (!$onchannels{$channel}) {
    &log("*** Cannot leave \#$channel. I'm not on that channel! ");
    return;
  }


  &logChannel($channel, "*** Leaving \#$channel ($msg)");

  $self->part("\#$channel", '$msg');



  &log("*** Closing log file for channel \#$channel") if ($config{debug});


  # bot is not there anymore...
  &removeChannel($channel);



}



#
# adds channel to list of bot's channels
#
sub addChannel {
  my ($channel) = @_;

  $onchannels{$channel} = time();
  &log("added '\#$channel'") if ($config{debug});
```



```
  if ($config{debug}) {
    &log("channels:");
    my $c;
    foreach $c (keys %onchannels) {
      &log("'$c', ");
    }
  }

}

#
# removes channel from list of bot's channels
#
sub removeChannel {
  my ($channel) = @_;

  delete $onchannels{$channel};
  delete $channelnicks{$channel};
  delete $opchannels{$channel};
  &log("removed '\#$channel'") if ($config{debug});

  delete $channelnicks{$channel};
  delete $opchannels{$channel};

  if ($channelnicks{$channel}) {
    &log("* I still have nicklist of $channel. FIX THIS!");
  }
  if ($opchannels{$channel}) {
    &log("* I still have oplist for $channel. FIX THIS!");
  }

  if ($onchannels{$channel}) {
    &log("!!! For some reason $channel is still on channels list !!!");
  }

  if ($config{debug}) {
    &log("channels:");
    my $c;
    foreach $c (keys %onchannels) {
      &log("'$c' $onchannels{$c}");
    }
  }

  if (!%onchannels) {
    # bot is not on any channel anymore :)
    &log("*** I'm not on any channel. Well, I will idle then and join to some channel later.");

    # check out number of known channels
    my $pptr = $arrays{'known_channels'};
    if ($#$pptr < 1) {
      &log("*** Unfortunatelly I don't know any channels to join! Please help!");
    }


  }

  # close that file handle
  my $chfh = &getFileHandle($channel);

  $conn->parent->queue(time + 5,
                       sub{
                         close $chfh || &log("*** Error closing filehandle '$chfh': $!");
                       }, $conn);
}



sub getFileHandle {
```

```
  my ($channel) = @_;

  my $secchannel = $channel;
  $secchannel =~ s/\/|\;|\&|\<|\>//g;

  my $chfh = "logfile_".$secchannel."_handle";

  return $chfh;
}



#
# join to $channel
# params: self, channelname, key
sub joinChannel {
  my ($self, $channel) = @_;


  if (!$self || !$channel) {
    &log("*** ERROR: not enough parameters for &joinChannel()") if ($config{debug});
    return;
  }

  # there might be a key in the channel name
  my $key = "";
  ($channel, $key) = split /\ /, $channel;

  $channel = lc($channel);

  # this is necessary for using scalar as filehandle!!!
  no strict "refs";


  # check if channel was forbidden
  my $chn = "";
  foreach $chn ($arrays{'dontgochannels'}) {
    $chn = lc($chn);
    if ($channel eq $chn) {
      &log("\#$channel is forbidden. Join cancelled.");
      return;
    }
  }

  # which log file to use
  my ($sec, $min, $hour, $day, $month, $year) = &getTime( localtime(time()) );
  my $secchannel = $channel;
  $secchannel =~ s/\///g;
  $LOGFILE = "$PUBLOGPATH/$secchannel.$year\_$month\_$day.txt";

  $channel = lc($channel);

  if ($channel =~ /^\.|^\;|^\&|^\//) {
    &log("*** ILLEGAL CHANNEL '$channel'");
    return;
  }

  &addChannel($channel);


  # this is for security
#  my $chfh = "logfile-$channel";
  my $chfh = &getFileHandle($channel);

  &log("*** Opening new log file '$LOGFILE'.") if ($config{debug});
  open ($chfh, ">>$LOGFILE") || die ("Can't open LOGIFILE: $LOGFILE: $!");

  if ($key) {
    &logChannel($channel, "*** Joining \#$channel with key");
```



```
  }
  else {
    &logChannel($channel, "*** Joining \#$channel");
  }

  if ($key) {
    # join channel with key
    $self->join("\#$channel", "$key");
  }
  else {
    # join channel
    $self->join("\#$channel");
  }



}



#
# return time in common format
#
sub getTime {
  my ($sec, $min, $hour, $day, $month, $year) = @_;

  if ($sec < 10) { $sec = "0$sec"; }
  if ($min < 10) { $min = "0$min"; }
  if ($hour < 10) { $hour = "0$hour"; }
  if ($day < 10) { $day = "0$day"; }
  if ($month < 10) { $month = "0$month"; }

  $month++;
  $year += 1900;

  # return the time
  return ($sec, $min, $hour, $day, $month, $year);
}




#
# get random channel
#
sub randChannel {
  my $pptr = $arrays{'known_channels'};
  my $max = $#$pptr;

  if ($max < 1) {
    &log("*** Unfortunatelly I don't know any channels to join! Please help!");
  }

  my $nr = int(rand($max)+1);
  my $chan = "";

  $chan = @$pptr[$nr];

  return $chan;
}


# What to do when someone got kicked
sub on_kick {
  my ($self, $event) = @_;
  my $oper = $event->nick;
  my ($to) = ($event->to)[1];
  my ($nick) = ($event->to)[0];
```

```
  my ($channel) = ($event->args)[0];

  my (@args) = ($event->args);
  my (@to) = ($event->to);

  my $userhost = $event->userhost;
  $channel =~ s/\#//g; # remove # from channel name

  shift (@args);

  if ($nick eq $bot_nick) {

    # if bot was kicked
    &logChannel($channel, "# I got kicked from \#$channel by '$oper' ($userhost) [@args].");

    # this is for log handling only
    &leaveChannel($self, $channel);


    # whine a little bit
    &msgPhrase($self, $oper, $arrays{'phrases_curse'});


  }
  else {

    # someone got kicked
    &logChannel($channel, "$nick got kicked from \#$channel by '$oper' ($userhost) [@args].");

    # say something funny about person just left
    if ($config{onleaveflamefreq} && &ifrand($config{onleaveflamefreq}, 'onleaveflam
efreq') ) {
      &sayPhrase($self, $channel, $nick, $arrays{'phrases_onleaves'});
    }

  }

}



# Handles some messages you get when you connect
sub on_init {
  my ($self, $event) = @_;
  my (@args) = ($event->args);
  shift (@args);

  &log("*** @args");
}

# on noprivileges; last action I did, I might have not been privileged to
# do it
sub on_noprivileges {
  my ($self, $event) = @_;
  my (@args) = ($event->args);
  shift (@args);

  &log("*** No privileges: @args");
}


#
# someone left
#
sub on_part {
  my ($self, $event) = @_;
  my ($channel) = ($event->to)[0];
  my $nick = $event->nick;
```



```
  $nick = lc($nick);
  my $userhost = $event->userhost;
  $channel =~ s/\#//g; # remove # from channel name

  &removeNick($channel, $nick); # remove user from nicklist on channel

  &logChannel($channel, "*** $nick ($userhost) has left channel ".$channel);

#  # check out number of people on channel, leave if boring
#  my $max = $#{$channelnicks{$channel}};
#  if ($max < 1 && $nick ne $bot_nick) {
#    &logChannel($channel, "*** It seems that I'm alone on this channel. I want
to go away!");
#    &leaveChannel($self, $channel);
#  }

  # say something funny about person just left
  if ($config{onleaveflamefreq} && &ifrand($config{onleaveflamefreq}, 'onleaveflamefr
eq') ) {
    &sayPhrase($self, $channel, $nick, $arrays{'phrases_onleaves'});
  }

}



#
# someone left
#
sub on_quit {
  my ($self, $event) = @_;
  my $nick = $event->nick;
  my $userhost = $event->userhost;
  my $reason = sprintf("%s", $event->args);
  my $to = sprintf("%s", $event->to);

  $nick = lc($nick);

  my $channel;
  my $chnick;

  &log("*** [$to] $nick ($userhost) has quit: '$reason' [".$event->args."]");

  #
  # as quit-event does not tell the channels, we need to check out them ourselve
s
  #

  # all channels
  foreach $channel (keys %onchannels) {

    &log ("** #$channel:") if ($config{debug});

    # all nicks on channel
    foreach $chnick (@{$channelnicks{$channel}}) {
      $chnick = lc($chnick);
      &log ("**** $chnick") if ($config{debug});

      # if nick was on that channel
      if ($nick eq $chnick) {

        &log ("****** $chnick == $nick ") if ($config{debug});

        # remove nick from channel list
        &removeNick($channel, $nick); # remove user from nicklist on channel

        # log event
        &logChannel($channel, "*** $nick ($userhost) has quit: '$reason'");

        # say something funny about person just left
```

```
        if ($config{onleaveflamefreq} && &ifrand($config{onleaveflamefreq}, 'onleaveflamefreq') ) {
          &sayPhrase($self, $channel, $nick, $arrays{'phrases_onleaves'});
        }

      }
    }
  }



}



sub on_whoisreply {
  my ($self, $event) = @_;
  my @args = $event->args();
  my $nick = $args[1];
  $nick = lc($nick);

  &log("** Whoisreply: @args") if ($config{debug});

#  # get last part of args
#  my $ircname = pop @args;
#
#   # split words
#  my (@words) = split /\ /, $ircname;

}



sub on_whoischannels {
  my ($self, $event) = @_;
  my @args = $event->args();
  my $me = shift @args;
  my $nick = shift @args;
#  my $nick = $args[1];
  $nick = lc($nick);
  $me = lc($me);

  &log("** Whoischannels: [$me] [$nick] @args") if ($config{debug});

  my @chans = split / /, $args[0];
  foreach my $chnn (@chans) {
    $chnn = lc($chnn); # lowercase

    &learnChannel($chnn, $arrays{'known_channels'});
  }

}



sub checkRealname {

  my ($self, $nick, $ircname) = @_;

   # split words
  my (@words) = split /\ /, $ircname;

    #
    # check realname
    #
    if ($#words < 1) {
      # need atleast 2 words
      &log("*** '$nick' ircname is too short: '$ircname'");
```



```
      my $channel;
      foreach $channel (keys %onchannels) {
        # for each channel the bot is on

        if ($opchannels{$channel}) {
          # check only channels the bot is an @
          my $chnick;
          $channel = lc($channel);
          foreach $chnick (@{$channelnicks{$channel}}) {
            # for each nick on that channel

            $chnick = lc($chnick);

            if ($config{realnamecheck}) {
              if ($nick eq $chnick) {
                &kickUser($self, $channel, $nick, "Mikä nimi 'ircname' muka on??");
                &msg($self, $nick, "Hei $nick! Kirjoita se oma nimesi IRC-clientissäsi siihen kohtaa missä sitä kysytään niin voit tulla \#$channel -kanavalle! 'ircname' ei taida olla oikea nimesi... ; ps. käy www.peelo.com:ssa!)");
              }
            }
          }
        } # if op

      }

    }
}



# What to do when someone joins a channel the bot is on.
sub on_join {
  my ($self, $event) = @_;
  my $nick = $event->nick;
  my $userhost = $event->userhost;
  my ($channel) = ($event->to)[0];

  $nick = lc($nick);
  my $in = "\\\\";
  if ($nick =~ /$in/g) {
    &log("*** Illegal nick: $nick");
    return;
  }

  $channel =~ s/\#//g; # remove # from channel name
  $channel = lc($channel);

  &addNick($channel, $nick); # add user to nicklist on channel

  &logChannel($channel, "*** $nick ($userhost) has joined channel \#$channel.");

  &whois($self, $nick) if ($nick ne $bot_nick);

}



#
# Checks giver userinfo agains given userlist
# returns 1 if found, 0 if not found.
# Prints debug if $config{debug} = 1
# Input: ('nick!user@host', @userlist)
#
sub checkuser {
  my ($us_nick, $uuser, @userlist) = @_;

  if (!$us_nick || !$uuser || !@userlist) {
    &log("# INVALID CALL OF &checkuser(). Not Enough arguments.");
```

```
  }

  # lowercases are simplier to handle
  $us_nick = lc($us_nick);
  $uuser = lc($uuser);

  my ($rest, $listed, $listed_nick, $listed_user, $listed_host) = "";
  my ($us_user, $us_host) = "";


  ($us_user, $us_host) = split /\@/, $uuser;

  # make non-* versions
  my $us_nick_p = $us_nick;
  my $us_user_p = $us_user;
  my $us_host_p = $us_host;
  $us_nick_p =~ s/\*|\^|\-|\~//; # get rid of foo characters
  $us_user_p =~ s/\*|\^|\-|\~//;
  $us_host_p =~ s/\*|\^|\-|\~//;

  my ($listed_nick_p, $listed_user_p, $listed_host_p) = "";

  # foreach operator on list
  foreach $listed (@userlist) {
    $listed = lc($listed); # lower case

    # split more and more and more
    ($listed_nick, $rest) = split /\!/, $listed;
    ($listed_user, $listed_host) = split /\@/, $rest;

    # make non-* versions
    $listed_nick_p = $listed_nick;
    $listed_user_p = $listed_user;
    $listed_host_p = $listed_host;
    $listed_nick_p =~ s/\*//; # get rid of *'s
    $listed_user_p =~ s/\*//;
    $listed_host_p =~ s/\*//;

    if (!$listed_nick || !$listed_user || !$listed_host) {
      &log ("\# ERROR: Invalid content in list: ('$listed_nick' '$listed_user' '$listed_host').");
    }
    if ($config{debug} > 3 ) {
      &log ("\# checking '$listed' ('$listed_nick', '$listed_user', '$listed_host') ");
      &log ("\tagainst ('$us_nick', '$us_user', '$us_host') ");
    }

    # nicely structured if's
    if (
        (($listed_nick =~ /\*/) && ($us_nick_p =~ /$listed_nick_p/gi)) ||
# '*nick*'
        (($listed_nick !~ /\*/) && ($listed_nick eq $us_nick)) ||       # 'nick'
        ($listed_nick eq '*')                                           # '*'
        )
      {
        # nick matched pattern
        &log ("\# Nick matched") if ($config{debug});

        if (
            (($listed_user =~ /\*/) && ($us_user_p =~ /$listed_user_p/gi)) ||
# '*user*'
            (($listed_user !~ /\*/) && ($listed_user eq $us_user)) ||
# 'user'
            ($listed_user eq '*')
# '*'
            )
          {
            # user matched pattern
            &log ( "\# User matched") if ($config{debug});
```



```
            if (
                (($listed_host =~ /\*/) && ($us_host_p =~ /$listed_host_p/gi)) |
| # '*host*'
                (($listed_host !~ /\*/) && ($listed_host eq $us_host)) ||
# 'host'
                ($listed_host eq '*')                                   # '*'
                )
              {
                # and host matched pattern
                &log ("\# Host matched") if ($config{debug});
                &log ("\# All matched. Returning 1.") if ($config{debug});

                 # all patterns matched!
                return 1;

              }
            else {
              &log ("\# Host mismatched") if ($config{debug});
            }

          }
        else {
          &log ("\# User mismatched") if ($config{debug});
        }
      }
    else {
      &log ("\# Nick mismatch.") if ($config{debug});
    }

  }  #foreach

  # this user did not match the pattern (maybe some parts of it, but not all)
  &log ("\# Did not match completely. Returning 0.") if ($config{debug});
  return 0;

}




# What to do when we receive a private PRIVMSG.
sub on_msg {
  my ($self, $event) = @_;
  my ($nick) = $event->nick;
  my $userhost = $event->userhost;
  my $msg = sprintf("%s", $event->args);

  $nick = lc($nick); # we use lowercase nicks all the time

  return if ($nick eq $bot_nick); # this HAS happened ;)

  # ignore short messages
  return if (length($msg) < 3);


  # check ignores
  foreach my $ignos ($arrays{'ignores'}) {
    if ($nick eq $ignos) {
      &logMessage($nick, "*** [IGNORED] <$nick> $msg");
      &msg($self, $nick, "Olet ignoressa. Käy katsomassa www.peelo.com.");
      return;
    }
  }


  # check for repeat
  if ($inmsghash{$nick} && $inmsghash{$nick} eq $msg) {
```

```
    &logMessage($nick, "*** $nick is repeating, ignoring this line");
    return;
  }
  else {
    $inmsghash{$nick} = $msg;
  }


  #
  # learn #channels
  #
  if ($msg =~ /\#/gi) {
    my $ncc = $msg;
    $ncc =~ s/lle|lla|\://gi; # remove -lle and -lla and :
    &learnChannel($ncc, $arrays{'known_channels'});
  }


  # if we have messaged to user
  if ($msghash{$nick}) {
    # clear that info because user replied to us
    $msghash{$nick} = "";
  }



  if ( &checkuser($nick, $userhost, @masters) ) {

    # we don't want to log masters messages
    &log ("*\$$nick* ($userhost) $msg");

    if ($msg =~ /^join\ \#(.*)/i) {
      # join given #channel

      &msg($self, $nick, "[ACTION] Joining channel \#$1");
      &log("*** $nick told me to join channel \#$1");

      &learnChannel("\#$1", $arrays{'known_channels'});

      &joinChannel($self, $1);

      return;
    }
    elsif ($msg =~ /^say\ \#(.*?)\ (.*)/i) {
      # say that phrase
      &say($self, $1, $2);
      return;
    }
    elsif ($msg =~ /^cmd\ (.*)/i) {
      # execute ANY command :) * This may be too dangerous!!
      if (!$1) {
        &log("no cmd!");
      }
      else {
        &log ("[CMD] '$1'");
        $self->sl($1);
      }
      return;
    }
    elsif ($msg =~ /^part\ \#(.*)/i) {
      # leave channel
      &msg($self, $nick, "[ACTION] Leaving channel \#$1");
      &log("*** $nick told me to leave channel \#$1");

      &leaveChannel($self, $1);
      return;
    }
    elsif ($msg =~ /^nick\ (.*)/gi) {
      # change nick
```



```
      &msg($self, $nick, "[ACTION] Changing nick to $1");
      &log("*** $nick told me to change nick to $1");

      $bot_nick = $1;
      $self->nick($1);
      return;
    }
    elsif ($msg =~ /^topic\ \#(.*?)\ (.*)/gi) {
      # change topic

      &msg($self, $nick, "[ACTION] Changing topic on channel \#$1");
      &log("*** $nick told me to change topic on channel \#$1 to $2");

      &changeTopic($self, $1, $2);
      return;
    }
    elsif ($msg =~ /^msg\ (.*?)\ (.*)/gi) {
      &msg($self, $1, $2);
      return;
    }
    elsif ($msg =~ /^kick\ \#(.*?)\ (.*)/gi) {
      # kick user out
      # channel, nick, reason
      my $ri =  &randomPhrase(undef, $arrays{'phrases_curse'});

      &kickUser($self, $1, $2, $ri);
      return;
    }
    elsif ($msg =~ /^ban\ \#(.*?)\ (.*)\ (.*)/gi) {
      # ban user from #$2 to $3 seconds
      &banUser($self, $1, $2, $3);
      return;
    }
    elsif ($msg =~ /^unban\ \#(.*?)\ (.*)/gi) {
      # unban
      &unbanUser($self, $1, $2);
      return;
    }
    elsif ($msg =~ /^newchan/gi) {
      my $nc = &randomKnownChannel();
      &log("Joining to $nc") if ($config{debug});
      &joinChannel($self, $nc);
    }
    elsif ($msg =~ /^status/gi) {
      # admin asked for STATUS

      # print all channels bot is on
      my ($xqchn, $xqchs);
      foreach $xqchn (keys %onchannels) {
        $xqchs .= "#$xqchn ";
      }

      my $uptime = time() - $time_start;

      &msg($self, $nick, "[STATUS] On channels: $xqchs. Uptime: $uptime seconds.");

    }
    elsif ($msg =~ /^move/gi) {
      # bot must move on to next channel

      &msg($self, $nick, "[ACTION] Joining new random channel...");
      &log("[ACTION] $nick asked me join new random channel");

      &joinChannel($self, &randChannel());
      return;
    }
    elsif ($msg =~ /^die/) {
      &msg($self, $nick, "[DIE] Oh my god, you killed me!");
```

```
      &quitter($self);
    }
    elsif ($msg =~ /^dontgo\ \#(.*)/) {
      # dont go to that channel anymore...
      my $pptr = $arrays{'dontgochannels'};
      push @$pptr, $1;
      &msg($self, $nick, "[SETTINGS] Added '$1' to ".$arrays{'dontgochannels'}.".");
      &log("[SETTINGS] $nick added '$1' to ".$arrays{'dontgochannels'});

      return;
    }
    elsif ($msg =~ /^ignore\ (.*)/) {
      # ignore user
      &ignoreUser($1);
      &msg($self, $nick, "[SETTINGS] Added '$1' to ".$arrays{'ignores'}.".");
      &log("[SETTINGS] $nick added '$1' to ".$arrays{'ignores'});

      return;
    }
    elsif ($msg =~ /^set (.*?) (.*)/g) {
      # SET configuration option (timeouts etc)

      &msg($self, $nick, "[CONFIG] $1=$2 [$config{$1}]");
      &log("[CONFIG] $1=$2 [$config{$1}]");
      $config{$1} = $2;
      return;

    }
    elsif ($msg =~ /^set/g) {
      # LIST configuration options and their values
      my $ct;
      &log("*** Configurations:");
      &msg($self, $nick, "[CONFIG] Listing configuration options:");

      foreach $ct (keys %config) {
        &log("\t$ct=$config{$ct}");
        &msg($self, $nick, "\t$ct=$config{$ct}");
      }

    }
    elsif ($msg =~ /^reboot/) {
      # master want's bot to reboot
      $KEEPRUNNING = 0; # this will be checked in main loop
      return;
    }
    else {

#       # learn all questions for later use ;)
#       if ($msg =~ /\?/) {
#        # take out nicks from beginning of phrases
#        my $narg = $msg;
#        $narg =~ s/^(.*?)\: //;
#        $narg =~ s/^(.*?)\, //;
#        $narg =~ s/^(.*?)\://;
#        $narg =~ s/^(.*?)\,//;
#        $narg =~ s/$bot_nick/\%nick/g;
#
#        &learnPhrase($narg, $arrays{'phrases_questions'});
#       }


      if (rand(100)+1 > 0) {

        &logMessage($nick, "*$nick* $msg");
        if (!$msg) {
          &log("EMPTY MESSAGE INPUT &onMsg");
          return;
        }
```



```
        my $rep = &getReply($msg);
        if ($rep) {
          &msgPhrase($self, $nick, $rep);
        }
        else {
          # we have no reply ;)
        }

      }


    }


  }
  else {

    &logMessage($nick, "*$nick* $msg");

    # learn all questions for later use ;)
    if ($msg =~ /\?/) {
      # take out nicks from beginning of phrases
      my $narg = $msg;
      $narg =~ s/^(.*?)\: //;
      $narg =~ s/^(.*?)\, //;
      $narg =~ s/^(.*?)\://;
      $narg =~ s/^(.*?)\,//;
      $narg =~ s/$bot_nick/\%nick/g;

      &learnPhrase($narg, $arrays{'phrases_questions'});
    }

    if (rand(100)+1 > 0) {

      if (!$msg) {
        &log("EMPTY MESSAGE INPUT &onMsg\n\n");
        return;
      }

      my $rep = &getReply($msg);
      if ($rep) {
        &msgPhrase($self, $nick, $rep);
      }
      else {
        # we have no reply ;)
      }

    }
  }


}



#
# send whois -command
#
sub whois {

  my ($self, $nick) = @_;

  &log("*** whois $nick") if ($config{debug});
  $self->sl("whois $nick");
}



sub on_public_handler {
```

```
  my ($self, $nick, $channel, $message) = @_;

  # we use only lowercase
  $channel = lc($channel);
  $nick = lc($nick);
  my $time = time(); # get current unixtime

  $onchannels{$channel} = $time; # set channels last public timestamp

  #
  # buffer all public lines
  #
  my ($oldnick, $oldtime, $oldmessage);

  my $pbsize = $#{$publicbuffer{$channel}}; # current size of this channels publ
ic buffer

  if ($#{$publicbuffer{$channel}} > $publicbuffersize) {
    # public buffer for this channel is full, take out first entry
    $oldmessage = pop @{$publicbuffer{$channel}};
    $oldtime = pop @{$publicbuffer{$channel}};
    $oldnick = pop @{$publicbuffer{$channel}};
    $pbsize--; # one removed, decrease this also
  }



  unshift @{$publicbuffer{$channel}}, ($nick, $time, $message);




  #
  # flood protection
  #
  my ($q, $w);

  my $floodrepeat = 0;
  my $floodrate = 0;

  for ($q = 0; $q <= $pbsize; $q += 3) {
    # check whole buffer

    if (@{$publicbuffer{$channel}}[$q] eq $nick) {
      # check all lines of this nick

      if(@{$publicbuffer{$channel}}[$q+2] eq $message) {
        # if user has made two similar lines

        $floodrepeat++;
      }

      if (@{$publicbuffer{$channel}}[$q+1] == $time) {
        # if user already said something on this same second
        $floodrate++;
      }

    }
  }

  if ($floodrepeat >= $config{floodrepeat}) {
    &logChannel($channel, "*** $nick is repeating atleast ". $config{floodrepeat} ." times. I will
kick and ban him.");
    &banUser($self, $channel, $nick, 60);
    &kickUser($self, $channel, $nick, "Älä toista!");
    return;
  }
  elsif ($floodrate >= $config{floodrate}) {
    &logChannel($channel, "*** $nick is flooding atleast $config{floodrate} lines per second. I will kick and
```



```
ban him.");
    &banUser($self, $channel, $nick, 120);
    &kickUser($self, $channel, $nick, "Älä floodaa!");
    return;
  }




  # check that bot is not messaging to ignored ones
  foreach my $ignos ($arrays{'ignores'}) {
    if ($nick eq $ignos) {
      &logChannel($channel, "[IGNORED] <$nick> $message");
      return;
    }
  }

  $chantalk{$channel} = $nick;

  $lasttalked = $nick;

  $time_public = time();
  my $onick = $nick;

  # $logarg is used to logging etc, $message for handling...
  my $logarg = $message;




  my @words = split /\ /, $message;
  my $word;


  # for all words on phrase
  foreach $word (@words) {

    # look for nicks
    my $nck;
    foreach $nck (@{$channelnicks{$channel}}){
      next if ($nck =~ /\]|\[|\\/); # some bug, those characters cannot be
                                    # handled correctly!?!

      if ($word =~ /$nck/i) {

        # that word was a nick on this channel
        &log("'$word' is a '$nck' on \#$channel") if ($config{debug});

        if (rand(100)+1 > 90) {
          # say something about that nick on channel
          &sayPhrase($conn,  $channel, $nck, $arrays{'phrases_questions'});
        }

        # modify original phrase; replace that with a tag for later use
        $message =~ s/$nck/\%randnick/gi;

        &log("* Replaced '$nck' with \%randnick") if ($config{debug});
      }
    }


#     # look for adjectives
#     my $adj;
#     foreach $adj (@words_adjectives){
#       next if (!$adj);
#       if ($word =~ /$adj/i) {
#         $message =~ s/$adj/\%adjective/gi;
#         &log("* '$word' is an adjective '$adj', replaced by tag \%adjective at &
```

```
on_public()");
#       }
#     }

    # look for numbers
    my $num;
    foreach $num ($arrays{'words_numbers'}){
      if ($word =~ /$num/i) {
        $message =~ s/$num/\%number/gi;
        &log("* '$word' is an number '$num', replaced by tag \%number at &on_public()");
      }
    }



  }

  ########



  # learn new adjective (if any)
  if ($message =~ /^(.*)\ on\ (.*)/gi ) {
    &log("* '$1' ON '$2'");
    my @ad = split /\ /, $2;
    my $adj = $ad[0];
    &log("\t*** Learnt adjective '$adj'");
    &learnPhrase($adj, $arrays{'words_adjectives'});
  }



  # learn all questions for later use ;)
  if ($message =~ /\?/) {
    # take out nicks from beginning of phrases
    my $narg = $message;
    $narg =~ s/^(.*?)\: //;
    $narg =~ s/^(.*?)\, //;
    $narg =~ s/$bot_nick/\%nick/g;

    &learnPhrase($narg, $arrays{'phrases_questions'});
  }



  # ban lamers ;)
  if ($message =~ /(antakaa|antaa|anna|saanko|saisko)(.*)(opit|oppi|opsit|obsit)
/gi) {
#     if (! &checkuser($nick, $userhost, @masters) ) {
      &banUser($self, $channel, $nick);

      my $ri = &randomPhrase(undef, $arrays{'phrases_curse'});
      &kickUser($self, $channel, $nick, $ri );
      &sayPhrase($self, $channel, $nick, $arrays{'phrases_curse'});

#     }
#     else {
#       &opUser($self, $channel, $nick);
#       return;
#     }
  }



  #
  # learn #channels
```



```
  #
  if ($message =~ /\#/gi) {
    $message =~ s/lle|lla//gi; # remove -lle and -lla
    &learnChannel($message, $arrays{'known_channels'});

    $message =~ s/(\#.*?) /\%randomchannel /g;
  }

  #
  # Convert nick a bit
  #

  if (rand(100)+1 < 30) {

    # lowercase
    $nick = lc($nick);

    # un-3l33th
    $nick =~ tr/0123456789/oi2e456789/;

    # take out non-word-characters
    $nick =~ s/\W//gi;
  }
  if (rand(100)+1 > 50)  {



      if (rand(100)+1 > 50)  {
        if (length($nick) > 6) {
          # take 4 first letters
          $nick = substr $nick, 0, length($nick)-3;
        }
      }
      else {
        if (length($nick) >= 5) {
          # take 5 first letters
          $nick = substr $nick, 0, length($nick)-1;
        }
      }

      # convert to lowercase
      $nick = lc($nick);


  }



  #
  # check out what to do
  #
  if ($message =~ /$bot_nick/gi)  {

    # asked to leave
    if ($message =~ /(mee\ |mene\ |menisitkö\ |menes\ )(pois|veke|muualle)/gi) {

      if (!$opchannels{$channel}) {
        # I'm asked to leave
        &logChannel($channel, "<$onick> $logarg");
#       &logChannel($channel, "*** $nick ($userhost) asked me to leave \#$channe
l, leaving.");
        &logChannel($channel, "*** $nick asked me to leave \#$channel, leaving.");
        &say($self, $channel, "$nick, tuu \#$bot_chan -kanavalle...");

        $self->parent->queue(time + 10,
                             sub{
                               &leaveChannel($self, $channel);
                             }, $self);
        &sayPhrase($self, $channel, $arrays{'phrases_curse'});
```

```
      }
      else {
        $self->parent->queue(time + 10,
                             sub{
                               &kickUser($self, $channel, $nick, "Mene itse! :)");
                             }, $self);
      }

      return;

    }
    else {
      &logChannel($channel, "#<$onick> $logarg");


      # get reply based on message
      &sayPhrase($self, $channel, $nick, &getReply($message));
    }


  }

  elsif ($config{nokeyreply} && &ifrand($config{nokeyreply}, 'nokeyreply') ) {

    &logChannel($channel, "<$onick> $logarg");
    &sayPhrase($self, $channel, $nick, &getReply($message));

  }
  else {
    # do nothing but log
    &logChannel($channel, "<$onick> $logarg");




  }

  # new
  &addNick($channel, $onick); # add user to nicklist on channel
}
############







#
# replies a reference to array of phrases
#
sub getReply {
  my ($message) = @_;



  if ($message =~ /tuli|takas|taas|kävit|sperm|lämmin|hieno|ilma|hauska|hassu/gi
) {
    return $arrays{'phrases_agrees'};
  }
  elsif ($message =~ /(kerro|osaatko)(.*)vitsi/gi) {
    return $arrays{'phrases_joke'};
  }
  elsif ($message =~ /(paljo|mitä|mita|onk(.*)sul)(.*)kello/gi) {
    my ($xsec, $xmin, $xhour, $xday, $xmonth, $xyear) = localtime(time());
    $xmonth++;

    $xmin = "0$xmin" if ($xmin < 10);
    $xsec = "0$xsec" if ($xsec < 10);
```



```
    # hours of day
    my @xhours = (
                  'kaksitoista', 'yksi', 'kaksi', 'kolme', 'neljä',
                  'viisi', 'kuusi', 'seitsemän', 'kahdeksan', 'yhdeksän',
                  'kymmenen', 'yksitoista',
                  'kaksitoista', 'yksi', 'kaksi', 'kolme', 'neljä',
                  'viisi', 'kuusi', 'seitsemän', 'kahdeksan', 'yhdeksän',
                  'kymmenen', 'yksitoista', 'kaksitoista'
                  );


    my @xtime = ();

    if ($xmin > 0 && $xmin < 15 ) {
      @xtime = "Vähän yli ".$xhours[$xhour] .".";
    }
    elsif ($xmin => 15 && $xmin < 30 ) {
      @xtime = "Taitaa olla $xhour:$xmin.";
    }
    elsif ($xmin == 30) {
      @xtime = "Se on puoli ".$xhours[$xhour+1].".";
    }
    elsif ($xmin > 30 && $xmin < 45) {
      @xtime = "Kohta viittätoista vaille ".$xhours[$xhour+1].".";
    }
    elsif ($xmin == 45) {
      @xtime = "Tasan viittätoista vaille ".$xhours[$xhour+1].".";
    }
    elsif ($xmin > 45 && $xmin <= 59) {
      @xtime = "Tulee kohta ".$xhours[$xhour+1].".";
    }
    elsif ($xmin == 0) {
      @xtime = "Aikamerkki kello ".$xhours[$xhour]. ". Seuraavaksi uutiset ja sää ;)";
    }
    else {
      @xtime = "Mun kello jäi kotiin, oisko kellään muulla?";
    }

    return \@xtime;

  }

# new ones here
  elsif ($message =~ /paljonko(.*)\?/i) {
    &learnPhrase($message, $arrays{'p_q_paljonko'});
    return $arrays{'p_a_paljonko'};
  }
  elsif ($message =~ /missä(.*)\?/i) {
    &learnPhrase($message, $arrays{'p_q_missa'});
    return $arrays{'p_a_missa'};
  }
  elsif ($message =~ /oletko(.*)\?/i) {
    &learnPhrase($message, $arrays{'p_q_oletko'});
    return $arrays{'p_a_oletko'};
  }


##
  elsif ($message =~ /botti|ihminen|human|elävä|botit|botte|ignore|ban|kick|kika
tk|potkika|windows|kernel|eggdrop|egge|eggi|mirc|ircii|mirk|scriini|kooda|bot|hu
oh|script|skript|robotti|koodi|linux|screen|ircii|browser|news|kovalevy|emolevy|
kampiakseli|ohjaustehostin|irkki|turing|kone|java|kooda|tekoäly|tekoaly|turink|3
1337|potti|ohjelma/g) {
    &learnPhrase($message, $arrays{'phrases_whatis'});
    return $arrays{'phrases_whatis'};
  }
  elsif ($message =~ /\%randomchannel\ /) {
    &learnPhrase($message, $arrays{'phrases_channels'});
```

```
    return $arrays{'phrases_channels'};
  }
  elsif ($message =~ /musa|hector|pyyhi\ kyynelee|alatalo|musiikki|tiktak|hei\ l
opeta|bombfunk|aikakone|music|hanson|poppi|stereo|mankka|hector|kaija\ koo|nylon
\ beat|cd|apulanta|euroviisu|kappale|piisi|biisi/gi) {
    &learnPhrase($message, $arrays{'phrases_mymusic'});
    return $arrays{'phrases_mymusic'};
  }
  elsif ($message =~ /teini|[redacted]|billu|määräil|maarail|pilde|bilde|flood|idioot
|juntti|juusto|peelo|p3310|haist|paska|haise|vedä|tyhmä|veda|tyhma|vittu|ämmä|vi
tu|vidu|perkele|saatana|pelle|[redacted]|[redacted]|kusipää|kahjo|bimbo|teini|juntti|juusto|
turpa|opsit|opit|helveti|\@/gi) {
```

Some strong Finnish profanities hidden

```
    return $arrays{'phrases_curse'};
  }
  elsif ($message =~ /tissi|panett|[redacted]
```

Finnish sexual verbs hidden before publishing source code

```
[redacted]|kanki|kanke/gi) {

    &learnPhrase($message, $arrays{'nosex'});

    return $arrays{'nosex'};
  }
  elsif ($message =~ /vanha\ |ikäs\ |vuosiker|vuotta|ikas\ |ikäin|ikäne|ikane|ik
ain|ikä\?|syntyny/gi) {
    &learnPhrase($message, $arrays{'phrases_myage'});
    return $arrays{'phrases_myage'};
  }
  elsif ($message =~ /kuka|nimi|kenen|nimes|mistä|mista|isä|äiti|asut|ketä/gi) {
    &learnPhrase($message, $arrays{'phrases_whoami'});
    return $arrays{'phrases_whoami'};
    }
  elsif ($message =~ /öitä|yötä|sänkyyn|nukkumaan|pehkuihin|petiin|goiaam|koisaa
|kuorsaamaan|nukuqsiin|nukq|älämölö|väsyt|nukut/gi) {
    &learnPhrase($message, $arrays{'phrases_goodnight'});
    return $arrays{'phrases_goodnight'};
  }
  elsif ($message =~ /anna|turpaa|valtaa|valtai|l337|mulle|miulle|minulle|antais
i|ottaisi|ottasi|antasi|mitat|vosu|huippu|hyvä|mahtava|upee|upea|hieno|siisti|to
ttakai|tietysti|aivan|totta|aina|anta|sä\ oot|ehdotan|iske|kimppuun|rule|tekoäl|
teoal|lysti|tuntuu|niinpä|niimpä|niinpa|niimpa/gi) {
    &learnPhrase($message, $arrays{'phrases_agrees'});
    return $arrays{'phrases_agrees'};
  }
  elsif ($message =~
    /mennää|onneks|onnex|kalaty|käläty|lähet|lahet|lähdetkö|sentää|sentaa|tuutko
|tuletko|kiinnostaa|kiinostaa|lahjoja|lahjoi|homii|hommii|jatka|lisää|tarkemmin|
selitä|aavan\ meren/gi) {
    return $arrays{'phrases_unknown'};
  }
  elsif ($message =~ /kuuluu|heittää|heittaa|pelata|pelaa|aloita|aloittaa|rukoil
e|pyydä|kerro|oletko|kertoi|olisit|poikakave|poitsufren|frendii|seuruste|muuta|s
tyyla|seurustel|kyllä|kylla|pannit|pannata|ban|kick|ignora/gi) {
    return $arrays{'phrases_deny'};
    }
  elsif ($message =~ /no\ niin|pompi|ravaa|haloo|kuulet|dallas|mene|pois|hiljaa|
ärsyttävä|älä|puhutaa|puhunu|jutellaa|jutellu|jutskaa|sano|sovitaan|pomppa|pompi
|lokiik|älyä|heitä|heita|puhua|läppää/gi) {
    &learnPhrase($message, $arrays{'phrases_ignorance'});
    return $arrays{'phrases_ignorance'};
  }
  elsif ($message =~ /tarkemmin|juttele|pompit|senssi|jutskaa|teit|juttele|kerro
|sano|laula|puhu|mulle|sinua|sinä|sina|minä|minua|mina|tylsä|kahjo|seko|hemmetin
/gi) {
    &learnPhrase($message, $arrays{'phrases_bored'});
    return $arrays{'phrases_bored'};
  }
  elsif ($message =~ /www\|http|weppi|webbi|veppi|url/gi) {
```



```
    &learnPhrase($message, $arrays{'phrases_bored'});
    return $arrays{'phrases_bored'};
  }
  elsif ($message =~ /moi|hei|moikka|heippa|tere|huhuu|juttuseuraa|tyttöj|pimuj|
terve|heps|iltaa|huomenta|päivää|morning|evening|paivaa|tere|moro|hello|hej|hi\
|mitäs/gi) {
    return $arrays{'phrases_greet'};
  }
  elsif ($message =~ /\?/) {
    # this seems to be good point to check if it was question ;)
    return $arrays{'phrases_answers'};
  }
  elsif ($message =~ /kysy|kerro|tiedä|muista|sano|laula|monta|paljon/) {
    # this seems to be good point to check if it was question ;)
    return $arrays{'phrases_questions'};
  }
  elsif ($message =~ /kuhan|oisko|kunhan|puhu|sano|jotain|mä\ vaan|kui\ |monta|m
eille|teille|oot|kiss|chat|ooks|oletko|ootsä|ootsa|ootko|missä|missa|miten|monta
k|milloin|mille|monelta|mistä|mista|mille|miltä|mitla|mihin|oox/gi) {
    &learnPhrase($message, $arrays{'phrases_wonders'});
    return $arrays{'phrases_wonders'};
  }
  else {
    return $arrays{'phrases_unknown'}; # unknown words, reply something stopid...
  } # if known words

}



# Prints the names of people in a channel when we enter.
sub on_names {
    my ($self, $event) = @_;
    my (@list, $channel) = ($event->args);    # eat yer heart out, mjd!
    $channel = lc($channel);


    # splice() only works on real arrays. Sigh.
    ($channel, @list) = splice @list, 2;

    $channel =~ s/\#//g; # remove # from channel name

    my $nlist = "@list";
    my (@nicks) = split/\ /, $nlist;

    my $nick;
#    foreach $nick (@nicks) {
#      next if ($nick eq $bot_nick);
#      $nick =~ s/\@|\+//g; # remove @ and +
#
#      &addNick($channel, $nick); # add user to nicklist on channel
#
#    }

    &logChannel($channel, "Users on $channel: @list");

#    # check out number of people on channel, leave if boring
#    my $max = $#{$channelnicks{$channel}};
#    if ($max < 1) {
#      &logChannel($channel, "*** It seems that I'm alone on this channel. I wan
t to go away!");
#      &leaveChannel($self, $channel);
#    }

}


# cfinger
sub on_finger {
```

```
    my ($self, $event) = @_;
    my $nick = $event->nick;

    $self->ctcp_reply($nick, join (' ', ($event->args), "$finger"));
    &log("*** CTCP FINGER request from $nick received");

}

# cuserinfo
sub on_userinfo {
    my ($self, $event) = @_;
    my $nick = $event->nick;

    $self->ctcp_reply($nick, join (' ', ($event->args), "$userinfo"));
    &log("*** CTCP USERINFO request from $nick received");
}


# cversion
sub on_version {
    my ($self, $event) = @_;
    my $nick = $event->nick;

    $self->ctcp_reply($nick, join (' ', ($event->args), "$version"));
    &log("*** CTCP VERSION request from $nick received");
}

# Yells about incoming CTCP PINGs.
sub on_ping {
    my ($self, $event) = @_;
    my $nick = $event->nick;

    $self->ctcp_reply($nick, join (' ', ($event->args)));
    &log("*** CTCP PING request from $nick received");

}

# Gives lag results for outgoing PINGs.
sub on_ping_reply {
    my ($self, $event) = @_;
    my ($args) = ($event->args)[1];
    my ($nick) = $event->nick;

    $args = time - $args;
    &log("*** CTCP PING reply from $nick: $args sec.");
}


# Change our nick if someone stole it.
sub on_nick_taken {
  my ($self) = shift;

  &log("*** Nick '$bot_nick' is reserved. Trying next nick on list...");

  sleep 5;

  my $newn = int(rand($#nicks)+1);
  my $bot_stolennick = $bot_nick;
  $bot_nick = $nicks[$newn];

  $self->nick($bot_nick);

  # inform that nick-taker that nick was reserved for me ;)
  $self->parent->queue(time + 2,
                       sub{
                         &msg($self, $bot_stolennick, "$bot_stolennick is my nick, please gi
ve it back! - $bot_stolennick on mun nikki, anna se heti takaisin!!!");
                       }, $self);
```



```
}



#
# What to do when we receive channel text.
#
sub on_public {
  my ($self, $event) = @_;
  my ($nick, $mynick) = ($event->nick, $self->nick);
  my ($message) = ($event->args);
  my $userhost = $event->userhost;
  my $res = "";
  my ($channel) = ($event->to)[0];
  $channel =~ s/\#//g; # remove # from channel name

  &on_public_handler($self, $nick, $channel, $message);
}


# Display formatted CTCP ACTIONs.
sub on_action {
  my ($self, $event) = @_;
  my ($nick, @args) = ($event->nick, $event->args);
  my ($dest) = ($event->to)[0];
  $dest =~ s/\#//g; # remove # from channel name

  # :FIXME: this should not be necessary:
  shift @args;


  if ($dest eq $bot_nick) {
    # it was private for bot
    &logMessage($nick, "*[CTCP ACTION] $nick @args");
    &on_public_handler($self, $nick, $nick, "@args");

  }
  else {
    my $channel = $dest;
    &logChannel($channel, "*[CTCP ACTION] $nick @args");
    &on_public_handler($self, $nick, $channel, "@args");

  }


  $time_public = time();

}



#
# when someone changes mode
#
sub on_mode {
  my ($self, $event) = @_;

  my (@to) = $event->to;
  my ($operhost) = $event->userhost;

  my ($oper, @args) = ($event->nick, $event->args);
  $oper = lc($oper);

  my $channel = $to[0];
  $channel = lc($channel);
  $channel =~ s/\#//g; # remove the hash

  my $mode = shift(@args);
```

```
my $target = shift(@args);

if ($oper eq $bot_nick && $oper eq $bot_nick && !$target) {
  # bot changed personal mode

  &log("*** I set personal mode $mode\n");
}
else {


  if (!$target) {
    # there was no target, it must have been to that channel
    $target = $channel;
  }

  $target = lc($target);

  &logChannel($channel, "*** $oper sets mode $mode $target on \#$channel");

  if ($target eq $bot_nick) {
    # mode has something to do with bot

    if ($mode =~ /\+b/) {
      # bot got banned

      &logChannel($channel, "*** I got banned by $oper!");

      if (!&checkuser($oper, $operhost, @masters)) {
        # if oper is not master, try to ban & nick him

        my $ri =  &randomPhrase($oper, $arrays{'phrases_curse'});
        &banUser($self, $channel, $oper);
        &kickUser($self, $channel, $oper, $ri );

      }

    }
    elsif ($mode =~ /\-b/) {
      # bot's ban was removed
      &logChannel($channel, "*** I got unbanned by $oper!");

    }
    elsif ($mode =~ /\+o/) {
      # bot got @
      &logChannel($channel, "*** I got @ from $oper!");
      $opchannels{$channel} = time();
    }
    elsif ($mode =~ /\-o/) {
      # bot lost @
      &logChannel($channel, "*** $oper took my @ out!");
      &sayPhrase($self, $channel, $oper, $arrays{'phrases_curse'});
      &msgPhrase($self, $oper, $arrays{'phrases_curse'});
      &msgPhrase($self, $oper, $arrays{'phrases_curse'});
      delete $opchannels{$channel};

    }
    elsif ($mode =~ /\-v/) {
      # bot lost voice
      &logChannel($channel, "*** $oper took my voice!");
      &msgPhrase($self, $oper, $arrays{'phrases_curse'});
    }
    elsif ($mode =~ /\+v/) {
      # bot got voice
      &logChannel($channel, "*** $oper gave me voice!");

    }

  } # to bot?
  else {
```



```
      # mode was for channel, not for the bot

      if ($mode =~ /\+m/) {
        # channel is moderated
        &logChannel($channel, "*** \#$channel is now moderated!");
      }
      elsif ($mode =~ /\-m/) {
        # channel is moderated
        &logChannel($channel, "*** \#$channel is not moderated anymore.");
      }

    }

  } # ELSE


}
#################




#
# other event
#
sub on_other {
  my ($self, $event) = @_;
  my @args = $event->args();

  &log("*** OTHER EVENT: @args");
}

#
# when bot cannot join to some channel for some reason we need to close the
# filehandle for that channel etc...
#
sub on_cantjoin {
  my ($self, $event) = @_;
  my @args = $event->args();
  my $channel = $args[1];
  $channel =~ s/\#//g;
  $channel = lc($channel);

  &logChannel($channel, "*** \#$channel $args[2]");

  &leaveChannel($self, $channel);
}



#
# 404 Cant Send to Channel
# leave the channel then
#
sub on_cantsendtochannel {
  my ($self, $event) = @_;
  my @args = $event->args();
  my $channel = $args[1];
  $channel =~ s/\#//g;
  $channel = lc($channel);

  &logChannel($channel, "*** \#$channel $args[2]");

  &leaveChannel($self, $channel);
}

#
# 482 You're not channel operator
# leave the channel then
```

```
#
sub on_youarenotchanop {
  my ($self, $event) = @_;
  my @args = $event->args();
  my $channel = $args[1];
  $channel =~ s/\#//g;
  $channel = lc($channel);

  &logChannel($channel, "*** \#$channel $args[2]");

}


#
# general error event...
#
sub on_error {
  my ($self, $event) = @_;
  my @args = $event->args();

  &log("*** ERROR EVENT: @args");

}


#
# someone invited bot
#
sub on_invite {
  my ($self, $event) = @_;

  my @args = $event->args();
  my $nick = $event->nick;
  my $nickhost = $event->userhost;
  my $to = $args[0];
  $to =~ s/\#//g;

  &log("*** '$nick' ($nickhost) invited me to channel \#$to");

  if ($to =~ /\//) {
    &log("** invalid channel name '$to'. Contains /");
    return;
  }
  if ($to =~ /\&/) {
    &log("** invalid channel name '$to'. Contains &");
    return;
  }

  if ($onchannels{$to}) {
    &log("l: $onchannels{$to}");
    &log("*** I'm already on \#$to. Perhaps I don't have to join again.");
    return;
  }

  &learnChannel("\#$to", $arrays{'known_channels'});

  &inviteUser($self, "$bot_chan", $nick);

  if ($config{joinoninvite}) {


    $self->parent->queue(time + 5,
                         sub{
                           &joinChannel($self, $to);
                           $self->parent->queue(time + 5,
                                                sub{
                                                  &say($self, $to, &randomPhrase(
$nick, $arrays{'phrases_oninvite'}) );
                                                }, $self);
```



```
                                                }, $self);



#  }
  }
  else {
    &msg($self, $nick, 'Hei! En pääse tulemaan juuri nyt, yritä myöhemmin uudelleen!');
  }


}


# Reconnect to the server when we die.
sub on_disconnect {
  my ($self, $event) = @_;
  my @args = ($event->args);

  &log("*** Disconnected from $bot_server: @args");

  sleep 1;
  $bot_server = $servers[int(rand($#servers)+1)];
  &log("*** Disconnected: ".$args[0]." , trying $bot_server");
  &do_connect();

}


sub on_nick {
  my ($self, $event) = @_;
  my ($nick, $mynick) = ($event->nick, $self->nick);
  my ($newnick) = ($event->args);
  my $userhost = $event->userhost;

  my $channel;
  my $chnick;

  my $newnicklog = $newnick;
  my $nicklog = $nick;

  $nick = lc($nick);
  $newnick = lc($newnick);


  # all channels
  foreach $channel (keys %onchannels) {

    # all nicks on channel
    foreach $chnick (@{$channelnicks{$channel}}) {

      $chnick = lc($chnick);

      # if nick was on that channel
      if ($nick eq $chnick) {

        # remove nick from channel list
        &removeNick($channel, $nick);

        # add new nick to channel
        &addNick($channel, $newnick);

        # log event
        &logChannel($channel, "*** $nicklog is now known is $newnicklog");

      }
    }
  }
```

```
}


# Look at the topic for a channel you join.
sub on_topic {
  my ($self, $event) = @_;
  my @args = $event->args();
  my $nick = $event->nick;
  my $nick_host = $event->from;
  my $channel;

  if ($event->type() eq 'notopic') {
    $channel = $args[1];
    $channel =~ s/\#//g;
    $channel = lc($channel);
    &logChannel($channel, "No topic set for \#$channel.\n");
  }
  elsif ($event->type() eq 'topic' and $args[0] ne $bot_nick) {
    # If it's being done _to_ the channel, it's a topic change.

    $channel = ($event->to)[0];
    $channel =~ s/\#//g;
  $channel = lc($channel);

    &logChannel($channel, "$nick ($nick_host) has changed the topic on channel \#$channel to $args[0]");
  }
  else {
    # it just is the topic
    $channel = $args[1];
    $channel =~ s/\#//g;
    $channel = lc($channel);

    &logChannel($channel, "The topic for \#$channel is $args[2]");
  }


}



sub do_connect {

  &log("*** Connecting to $bot_server");

  $conn = $irc->newconn(Server    => ("$bot_server"),
                        Port      => 6667,
                        Nick      => ("$bot_nick"),
                      Ircname   => ("$bot_realname"),
                        Username => ("$bot_username")
                        )
    or die "irctest: Can't connect to IRC server.\n";

}



#
# returns 1 or 0, rand from 0 to 1000 must be bigger than given freq
#
sub ifrand {
  # 0-1000
  my ($freq, $purpose) = @_;
  my $rand = int(rand(1000)+1);

  $purpose = "FIX::unknown!" if (!$purpose);
  return 0 if (!$freq);
  if ( $config{debug} > 5 ) {
```



```
    &log("RAND: $rand FREQ: $freq ($purpose)");
  }

  if  ( $rand < $freq) {
    return 1;
  }
  else {
    return 0;
  }
}




#
# ^C
#
sub quitter {
  my $self = shift;

  &log("*** Quitting as requested.");

  &do_quit();

  &log("Doing untie for arrays...");

  &untieAllArrays();


  &log("All saved.");

  &log("Removing pid-file...");
  unlink "$pidfile" || &log("Can't remove PIDFILE $pidfile: $!");
  &log("...removed ok.");

  &log("*** Now exiting. Bye.");
  exit;
}









#
# MAIN
#



#
# Start it
#



&log("*** Starting...");

while (1) {
  # this is not the main loop...

  $irc = new Net::IRC;
```

```
  &do_connect();


# tähän pitäis kirjoittaa temporary handler routinesysteemi, esim
#whoisreplyjen nappaamiseen!
#
#




  &log("*** Installing handler routines...");

  $conn->add_handler('nick',    \&on_nick);
  $conn->add_handler('cping',   \&on_ping);
  $conn->add_handler('crping', \&on_ping_reply);
  $conn->add_handler('msg',     \&on_msg);
  $conn->add_handler('public', \&on_public);
  $conn->add_handler('caction', \&on_action);
  $conn->add_handler('join',    \&on_join);
  $conn->add_handler('part',    \&on_part);
  $conn->add_handler('leave',    \&on_part);
  $conn->add_handler('quit',    \&on_quit);
  $conn->add_handler('sign',    \&on_quit);
  $conn->add_handler('topic',    \&on_topic);
  $conn->add_handler('notopic',    \&on_topic);




  $conn->add_global_handler([ 251,252,253,254,302,255 ], \&on_init);
  $conn->add_global_handler('disconnect', \&on_disconnect);
  $conn->add_global_handler(376, \&on_connect);
  $conn->add_global_handler(433, \&on_nick_taken);
  $conn->add_global_handler(353, \&on_names);

  $conn->add_global_handler('cfinger', \&on_finger);
  $conn->add_global_handler('cuserinfo', \&on_userinfo);
  $conn->add_global_handler('cversion', \&on_version);

  $conn->add_global_handler('mode', \&on_mode);
  $conn->add_global_handler('kick', \&on_kick);

  $conn->add_global_handler('noprivileges', \&on_noprivileges);

  $conn->add_global_handler('invite', \&on_invite);

  $conn->add_global_handler(401, \&on_other); # no such nick / channel

  $conn->add_global_handler('other', \&on_other);


  $conn->add_global_handler(311, \&on_whoisreply); # whoisuser
  $conn->add_global_handler(312, \&on_whoisreply); # whoisserver
  $conn->add_global_handler(313, \&on_whoisreply); # whoisoperator
  $conn->add_global_handler(314, \&on_whoisreply); # whoiswasuser
  $conn->add_global_handler(315, \&on_whoisreply); # endofwho
  $conn->add_global_handler(316, \&on_whoisreply); # whoischanop
  $conn->add_global_handler(317, \&on_whoisreply); # whoisisidle
  $conn->add_global_handler(318, \&on_whoisreply); # endofwhois
  $conn->add_global_handler(319, \&on_whoischannels); # whoischannels


  $conn->add_global_handler('error', \&on_error);

  $conn->add_global_handler(475, \&on_cantjoin); # +k
  $conn->add_global_handler(474, \&on_cantjoin); # +b
  $conn->add_global_handler(473, \&on_cantjoin); # +i
```



```
  $conn->add_global_handler(404, \&on_cantsendtochannel);

  $conn->add_global_handler(482, \&on_youarenotchanop); # you are not @




  $irc->do_one_loop();

  sleep 5; # waiting all things to start...

  $KEEPRUNNING = 1;

  my $Seconds_10 = 10; # 10..0, then reset
  my $Seconds_60 = 60; # 60..0



  while ($KEEPRUNNING) {
    # THIS is the main loop


    # do one loop, first round does the magic
    $irc->do_one_loop();

    $Seconds_10--;
    $Seconds_60--;

    if ($Seconds_10 <= 0) { $Seconds_10 = 10; }
    if ($Seconds_60 <= 0) { $Seconds_60 = 60; }

    # randomly join some known channel
    if ($config{autojoinfreq} && &ifrand($config{autojoinfreq}, 'autojoinfreq') ) {
      &log("Thinking about to join some random channel...") if ($config{debug});
      # join new channel
      &joinChannel($conn, &randomKnownChannel() );
    }
    elsif ( $config{automonologue} && &ifrand($config{automonologue}, 'automonologue
') ) {
      # say something to channel

      my $ch = &randomChannel;
      &log("Thinking about to say something randomly at '#$ch'") if ($config{debug});

      if ( $chantalk{$ch} && $chantalk{$ch} ne $bot_nick ) {
        # if bot was not last talker at channel... talk!
        &sayPhrase($conn,  $ch, undef, $arrays{'phrases_questions'});
      }
      else {
        if ($config{debug}) {
          &log("Skipping idea because I was last talker.");
        }
      }
    }
    elsif ( $config{autodialog} && &ifrand($config{autodialog}, 'autodialog') ) {
      # say something to someone

      my $ch = &randomChannel;
      my $nick = &randomNick($ch);

      if ($nick) {
        &log("Thinking about to say something to '$nick' randomly at '#$ch'") if ($config{debug});

        if ( $chantalk{$ch} && $chantalk{$ch} ne $bot_nick ) {
        # if bot was not last talker at channel... talk!
          &sayPhrase($conn,  $ch, $nick, $arrays{'phrases_questions'});
        }
```

```
        else {
          if ($config{debug}) {
            &log("\tSkipping idea because I was last talker.");
          }
        }
      }
    }
    elsif ( $config{autoinvite} && &ifrand($config{autoinvite}, 'autoinvite') ) {
    # invite random user from random channel to #$bot_chan
      my $ch = &randomChannel;

      if ($ch eq $bot_chan) {
        &log("*** I don't want to invite anyone from \#$bot_chan as he already is there ;)") if ($config{d
ebug});
      }
      else {
        my $nick = &randomNick($ch);
        if ($nick) {
          &log("Thinking about to invite '$nick' from '\#$ch' to '\#$bot_chan'") if ($config{debug})
;
          &inviteUser($conn, "$bot_chan", $nick);
        }
      }
    }
    elsif ($config{autotopic} && &ifrand($config{autotopic}, 'autotopic') ) {
      # change topic randomly
      my $ch = &randomChannel;
      &log("About to changing topic randomly at '#$ch'") if ($config{debug});
      if ($opchannels{$ch}) {
        &changeTopic($conn, $ch, &randomPhrase(&randomNick($ch), $arrays{'phrases_ques
tions'}) );
      }
      else {
        &log("Skipped idea: I'm not op on that channel!");
      }

    }



    #
    # ONCE PER MINUTE -timer
    #
    if ($Seconds_60 == 1) {



      # Leave idle channels, check all channels once per minute
      &log ("Testing idlelimit $config{idlelimit} on channels: ") if ($config{debug});
      # hits once per minute

      my $channel = ();
      foreach $channel (keys %onchannels) {
        &log("'\#$channel', ") if ($config{debug});

        # don't leave homechannel
        if ($channel eq $bot_chan) {
          &log(" (Ignoring homechannel '\#$channel') ") if ($config{debug});
          next;
        }

        # check other channels
        if ( time() - ($onchannels{$channel}) > $config{idlelimit}) {
          &log ("\nIdlelimit on channel '\#$channel'. Leaving...") if ($config{debug});
          &leaveChannel($conn, $channel);
        }

      }
```



```
      &log("-------------------------------------------------------");
    }



  } # while

  &log ("*** Bot is about to reboot, please wait!");

  #
  # If we get to this point, it means that something has turned KEEPRUNNING
  # to 0.
  # We just kind of re-boot ourselves

  $KEEPRUNNING = 1; # start running again

  &do_quit;

} # main loop


##
# Close consolelog
close CONSOLELOG;


#
##


##########
##########
#########
```