

Meta

- There's a rigid template for tech talks that I will attempt to follow
- I have a problem
- I can use perl to solve this problem!
- Here are some existing ways of doing it
- Here is a rough comparison (and they all suck, apart from..)
- Here's the way I [used / know most about / wrote]
- Examples (I'm so clever)

I have a problem

- I spend far too much time on IRC
- In fact, it's my main social wosssname
- I like my life to be easy
- I write IRC bots to make my life easier
- And annoy people at work
- pasty would be a good example
- namer is a better example
- as are slavorg / clunker

I can use perl to solve this problem!

- Naturally, I can write IRC bots in perl
- You can probably write IRC bots in postscript
- But this is not London.ps. Thankfully.

Here are some existing ways..

- IO::Socket
- Net::IRC
- POE::Component::IRC
- POE::Component::IRC::Object
- Bot::Pluggable
- IRC::Bot
- mozbot
- infobot

Rough comparison - IO::Socket

- Sure, if you like that sort of thing

```
#!/usr/bin/perl
use IO::Socket;
$sock = IO::Socket::INET->new(
    PeerAddr => 'server', PeerPort => 6667, Proto => 'tcp'
) or die "could not make the connection";

while($line = <$sock>){
    print $line;
    if($line =~ /(NOTICE AUTH).*(checking ident)/i){
        print $sock "NICK b0ilersbot\nUSER bot 0 0 :just a bot\n";
        last;
    }
}

while($line = <$sock>){
    print $line;
    #use next line if the server asks for a ping
    if($line =~ /^PING/){
        print $sock "PONG :" . (split(/ :/, $line))[1];
    }
}
```

Rough comparison - Net::IRC

- Not much more than 'read line from server, call the right function'
- Still lots of things you need to do yourself

```
use Net::IRC;
```

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

```
$conn = $irc->newconn(Nick    => 'some_nick',  
                    Server => 'some.irc.server.com');
```

```
$irc->start;
```

```
sub on_connect {  
    my $self = shift;  
    print "Joining #IRC.pm...";  
    $self->join("#IRC.pm");  
    $self->privmsg("#IRC.pm", "Hi there.");  
}
```

POE::Component::IRC

- Similar to Net::IRC
- Downside - uses POE
- I don't like POE any more
- Lots of setup needed, but nice after that
- Main benefit - uses POE
- Also POE::Component::IRC::Object

Bot::Pluggable

- Based on PoCo::IRC::Object
- Extensible bot framework
- You write modules, each module gets a shot at handling events
- Not really the simple framework I was after

IRC::Bot

- Single purpose bot
- Does several channel maintenance things
- not (easily) extensible

mozbot

- By the mozilla people
- No, I don't know why
- Based on Net::IRC
- Current CVS is 3090 lines
- Comes with 7575 lines of module code
- Does everything you could ever want

infobot

- dipsy!
- It's modular..
- ..but not in a very modern perl way
- current development version doesn't do IRC any more

Here's the one I inherited

- Bot::BasicBot
- It took 12 slides to get here!
- Written by Mark Fowler
- Based on POE::Component::IRC
- Taken over by Simon Batistoni when Mark got bored
- Taken over by me because I foolishly volunteered
- I've actually done some releases!

Design intent

- You don't have to know anything about IRC beyond normal user knowledge
- You don't need to understand POE
- You can use POE if you really want to
- Make easy things easy, and hard things possible
- Hide boring stuff as much as possible

Clever things it does

- decent reconnect
- 'tick' events for regular activity
- can fork a process and capture it's output
- understands direct addressing
- allows access to POE if you want it

Usage

- Subclass Bot::BasicBot
- Override methods for the events you want to handle
- Create new instance of your module, and call run

```
package MyBot;
use base qw(Bot::BasicBot);

sub said {
    my ($self, $message) = @_;
    my $who = $message->{who};
    return "Lalalala I can't hear you, $who..";
}

my $bot = MyBot->new( channels => [ '#jerakeen' ],
                    nick      => 'annoyingbot' );

$bot->run();
```

Optionally

- Implement `help()` method

```
sub help { "I refuse to talk to anyone. They're all boring." }
```

- Regular events

```
sub tick {  
    $self->say( channel => '#london.pm',  
               body     => "it's now ".localtime(time) );  
    return 10;  
}
```

- People leaving and joining channel

```
sub chanjoin {  
    my ($self, $message) = shift;  
    $self->reply($message,  
                "Welcome, $message->{who}. I'm annoying!");  
}
```

Bots I have written - namer

- Displays the title of any url mentioned in channel

```
package Namer;
use base qw(Bot::BasicBot);
use URI::Title qw(title);
use URI::Find::Simple qw(list_uris);

sub said {
    my ($self, $message) = @_;
    my $body = $message->{body};
    my @titles = map { title($_) } list_uris($message->{body});
    return join(" ", map( { "[ $_ ]" } @titles ));
}

Namer->new(
    channels => [ '#fotango', '#ponie' ],
    server => 'london.irc.perl.org',
    nick => 'namer',
)->run();
```

Bots I have written - file_watcher

- Displays changes to the files in a folder

```
sub tick {
  my $self = shift;
  my ($added, $removed) = $self->compare_files;
  for my $file (@$removed) {
    $self->say( channel => $chan, body => "removed $file" );
  }
  for my $file (@$added) {
    $self->say( channel => $chan, body => "added $file" );
  }
  return 5; # be called again in 5 seconds
}
```

Bots I have written - countdownbot

- Displays the time remaining until an event

```
sub tick {
  my $self = shift;
  # How long till the event?
  my $secs = Date::Parse::str2time($self->{date}) - time;

  $self->say( channel => '#fotango', body => from_now($secs) );

  # wait depending on how long is left
  if      ( $secs > 60 * 20 ) { return 60 * 10
  } elsif ( $secs > 60 ) {      return 60
  } elsif ( $secs > 10 ) {      return 10
  } elsif ( $secs > 0 ) {       return 1
  } else {                      exit; # done.
  }
}
```

Bots others have written - tailbot

- Tails a file to channel

```
use strict;
use base 'Bot::BasicBot';

sub connected {
    my $self = shift;
    $self->forkit({
        channel => '#tailbot',
        run      => [ qw ( /usr/bin/tail -f logfile ) ]
    });
}

main->new( nick => 'tailbot', channels => ['#tailbot'] )->run;
```

Further stuff

- You can easily turn a `Bot::BasicBot` bot into a `Bot::BasicBot::Pluggable` module

```
package Bot::BasicBot::Pluggable::Module::Title;
use base qw(Bot::BasicBot::Pluggable::Module::Base);
use URI::Title qw(title);
use URI::Find::Simple qw(list_uris);

sub said {
    my ($self, $mess, $pri) = @_;
    return unless ($pri == 0);

    for my $uri ( list_uris($mess->{body}) ) {
        $self->reply($mess, "[ ".title($uri)." ]") if title($uri);
    }
    return;
}
1;
```

Future plans

- messages should be objects
- it should be easier to turn a `Bot::BasicBot` into a pluggable module
- support for some of the simpler POE tasks
- maintain a list of people in a channel and their state
- tests!

More information

- <http://jerakeen.org/programming/Bot-BasicBot>
- <http://search.cpan.org/perl/doc?Bot::BasicBot>
- Thank you