SSH LIBRARY

SECURE SHELL

HISTORY

- V 1.X
- V 2.x (secsh)
- OpenSSH and OSSH

SECURE SHELL AND MORE

- Connection forwarder
- Transport protocol (Git, SFTP, SCP)
- Force remote command execution

HOW DOES SSH WORK

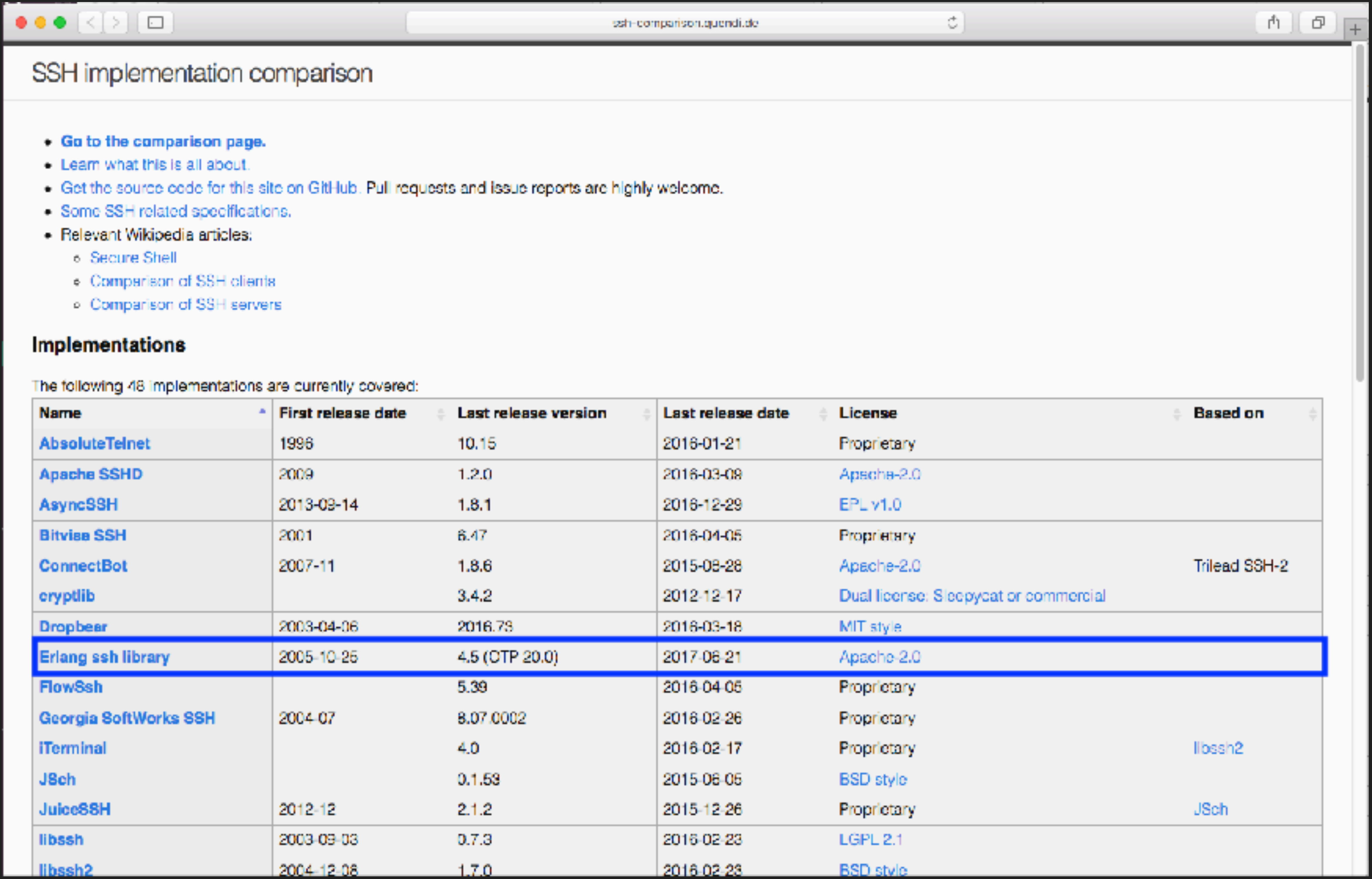
Public key infrastructure

Client

Server

https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process

ALL SSH IMPLEMENTATIONS


ssh-comparison.quendi.de

SSH implementation comparison

- **Go to the comparison page.**
- Learn what this is all about.
- Get the source code for this site on GitHub. Pull requests and issue reports are highly welcome.
- Some SSH related specifications.
- Relevant Wikipedia articles:
  - Secure Shell
  - Comparison of SSH clients
  - Comparison of SSH servers

Implementations

The following 48 implementations are currently covered:

Name	First release date	Last release version	Last release date	License	Based on
AbsoluteTelnet	1996	10.15	2016-01-21	Proprietary	
Apache SSHD	2009	1.2.0	2016-03-08	Apache-2.0	
AsyncSSH	2013-09-14	1.8.1	2016-12-29	EPL v1.0	
Bitvise SSH	2001	6.47	2016-04-05	Proprietary	
ConnectBot	2007-11	1.8.6	2015-08-28	Apache-2.0	Trilead SSH-2
cryptlib		3.4.2	2012-12-17	Dual license: Sleepycat or commercial	
Dropbear	2003-04-06	2016.73	2016-03-18	MIT style	
Erlang ssh library	2005-10-25	4.5 (OTP 20.0)	2017-06-21	Apache-2.0	
FlowSsh		5.39	2016-04-05	Proprietary	
Georgia SoftWorks SSH	2004-07	8.07.0002	2016-02-26	Proprietary	
iTerminal		4.0	2016-02-17	Proprietary	libssh2
JSch		0.1.53	2015-06-05	BSD style	
JuiceSSH	2012-12	2.1.2	2015-12-26	Proprietary	JSch
libssh	2003-09-03	0.7.3	2016-02-23	LGPL 2.1	
libssh2	2004-12-08	1.7.0	2016-02-23	BSD style	

ERLANG SSH MODULE

http://erlang.org/doc/man/ssh.html

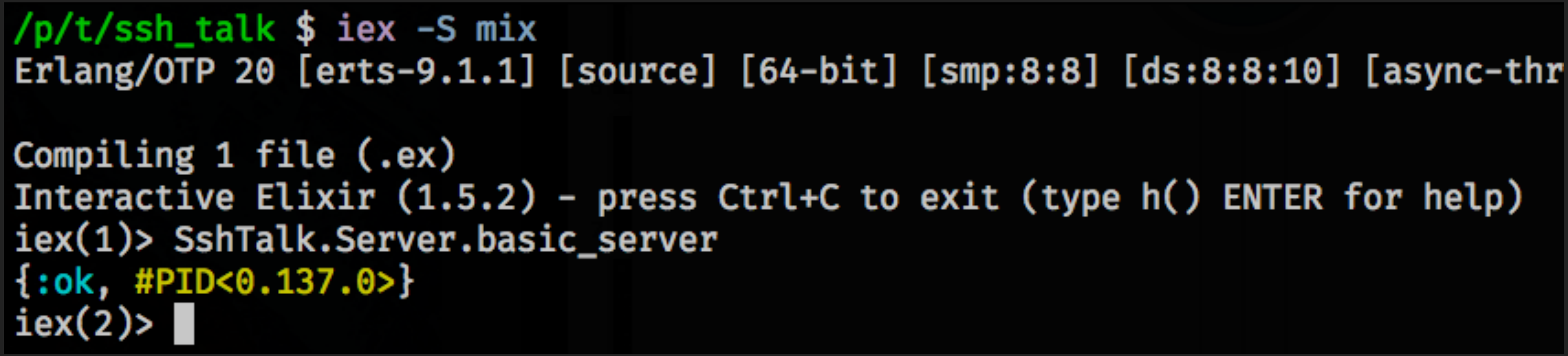
ERLANG SSH

- SSH Server
  - Shell, CLI, Password, Key callbacks
  - Load of configs
  - Custom Logging events
- SSH Client
  - Accept Host, Key callbacks
  - Load of configs
  - Custom Logging events

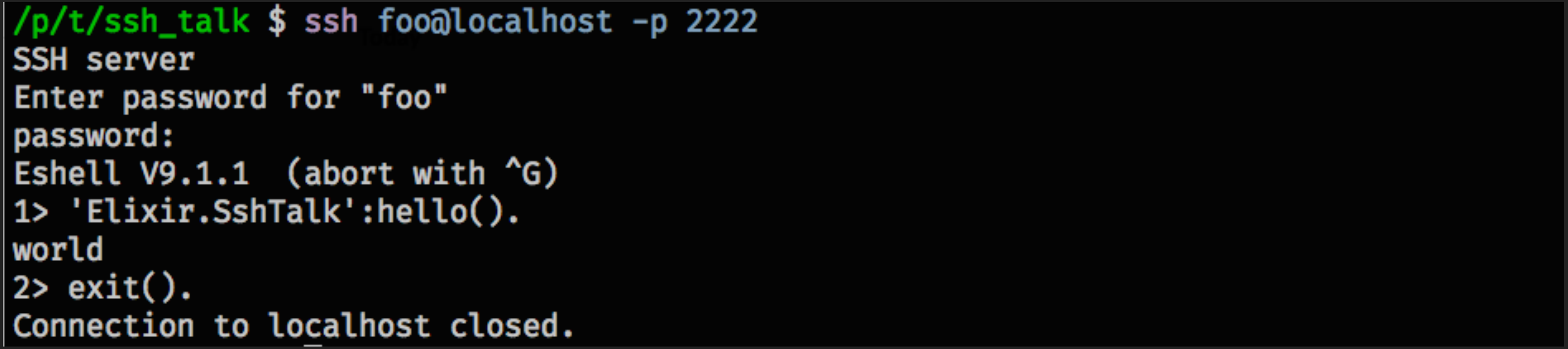
SSH SERVER

BASIC SSH SERVER AUTH WITH PASSWORD

```
def basic_server do
  :ssh.daemon 2222,
    system_dir: String.to_charlist(@root_dir),
    user_passwords: [{'foo', 'bar'}]
end
```

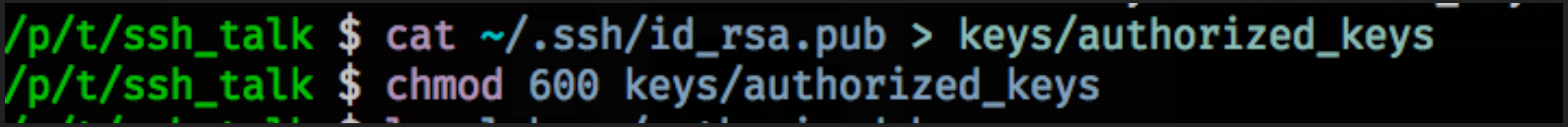


```
/p/t/ssh_talk $ iex -S mix
Erlang/OTP 20 [erts-9.1.1] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-thr

Compiling 1 file (.ex)
Interactive Elixir (1.5.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> SshTalk.Server.basic_server
{:ok, #PID<0.137.0>}
iex(2)>
```

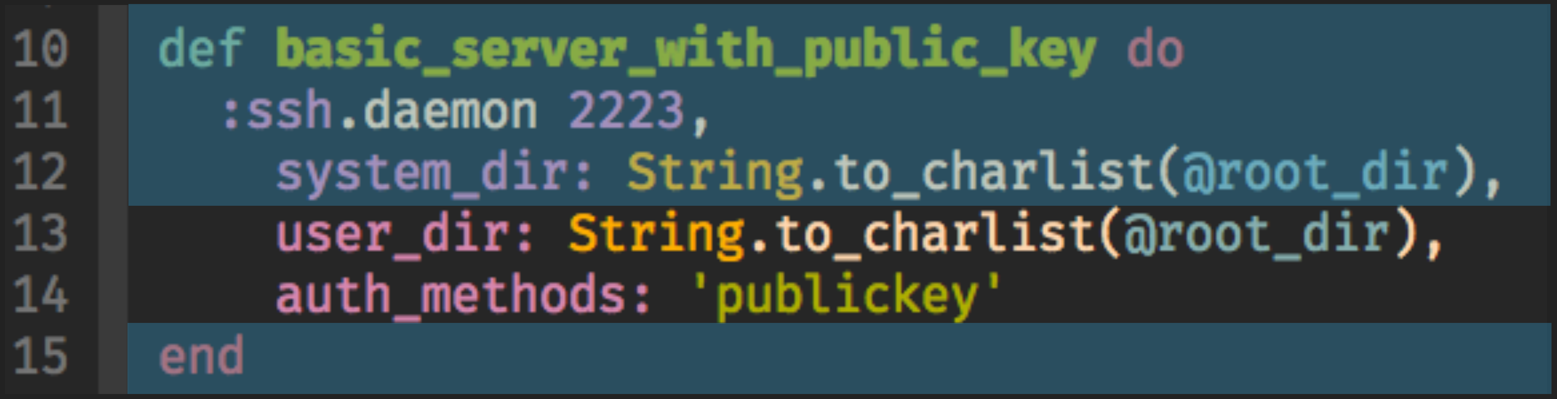


```
/p/t/ssh_talk $ ssh foo@localhost -p 2222
SSH server
Enter password for "foo"
password:
Eshell V9.1.1  (abort with ^G)
1> 'Elixir.SshTalk':hello().
world
2> exit().
Connection to localhost closed.
```

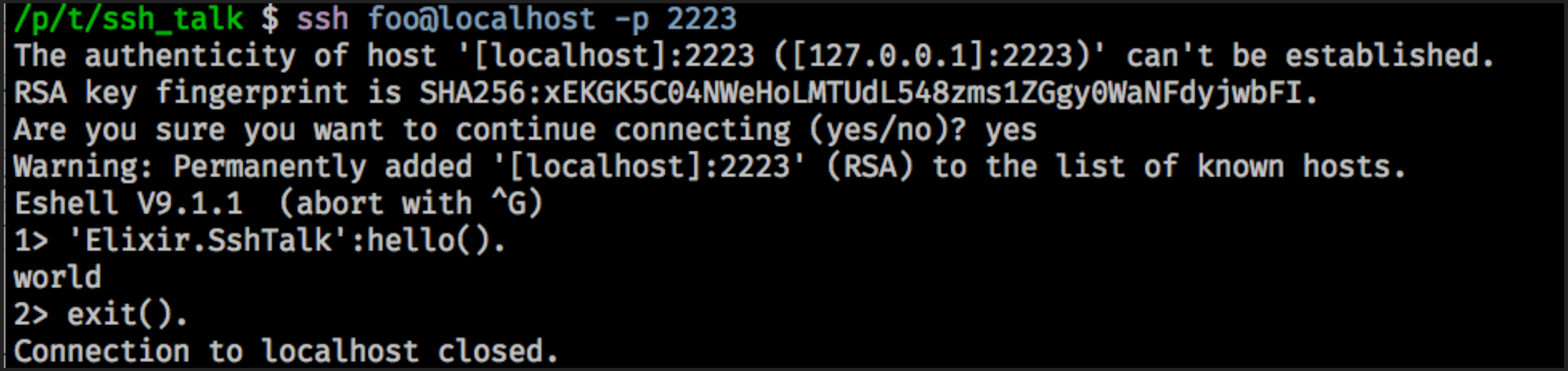
BASIC SSH SERVER AUTH WITH PUBLIC KEY



```
/p/t/ssh_talk $ cat ~/.ssh/id_rsa.pub > keys/authorized_keys
/p/t/ssh_talk $ chmod 600 keys/authorized_keys
```

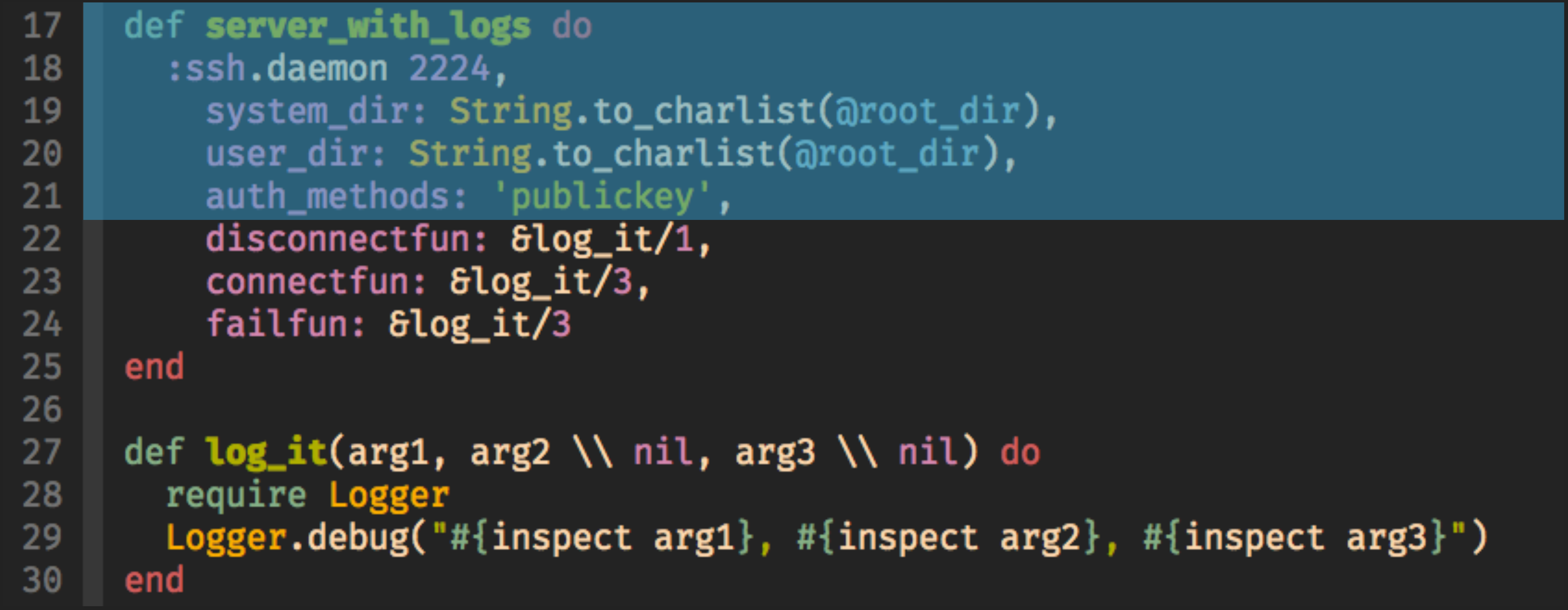


```elixir
def basic_server_with_public_key do
  :ssh.daemon 2223,
    system_dir: String.to_charlist(@root_dir),
    user_dir: String.to_charlist(@root_dir),
    auth_methods: 'publickey'
end
```

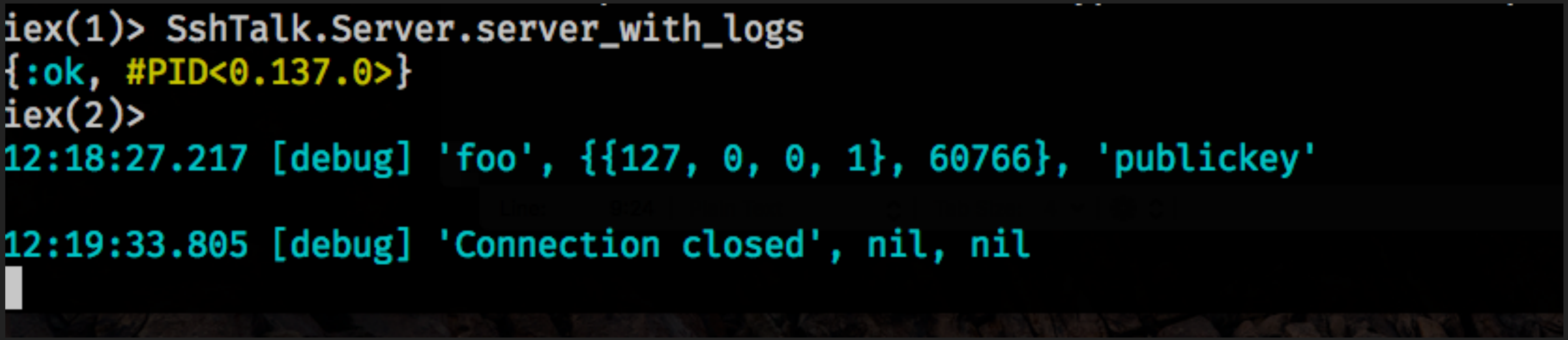


```
/p/t/ssh_talk $ ssh foo@localhost -p 2223
The authenticity of host '[localhost]:2223 ([127.0.0.1]:2223)' can't be established.
RSA key fingerprint is SHA256:xEKGK5C04NWeHoLMTUdL548zms1ZGgy0WaNFdyjwbFI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2223' (RSA) to the list of known hosts.
Eshell V9.1.1  (abort with ^G)
1> 'Elixir.SshTalk':hello().
world
2> exit().
Connection to localhost closed.
```

BASIC SSH WITH LOGGING

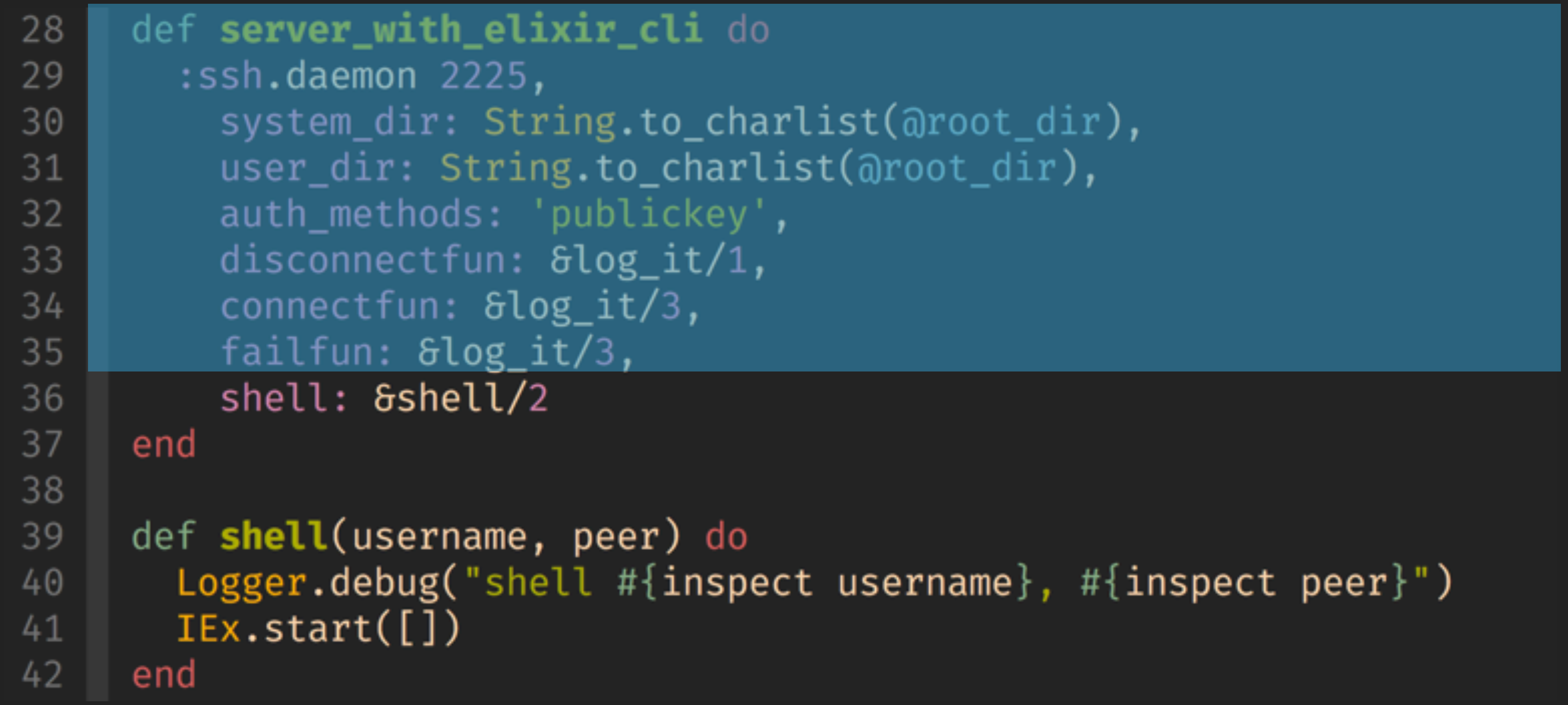


```
def server_with_logs do
  :ssh.daemon 2224,
    system_dir: String.to_charlist(@root_dir),
    user_dir: String.to_charlist(@root_dir),
    auth_methods: 'publickey',
    disconnectfun: &log_it/1,
    connectfun: &log_it/3,
    failfun: &log_it/3
end

def log_it(arg1, arg2 \\ nil, arg3 \\ nil) do
  require Logger
  Logger.debug("#{inspect arg1}, #{inspect arg2}, #{inspect arg3}")
end
```

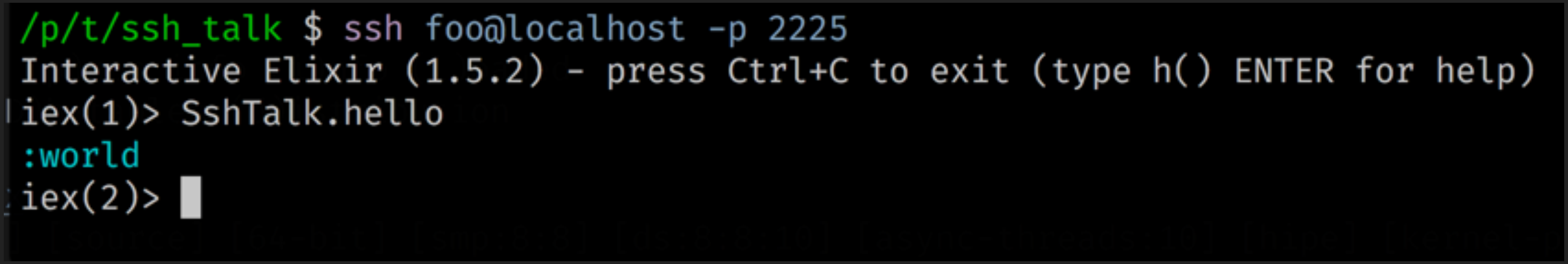


```
iex(1)> SshTalk.Server.server_with_logs
{:ok, #PID<0.137.0>}
iex(2)>
12:18:27.217 [debug] 'foo', {{127, 0, 0, 1}, 60766}, 'publickey'

12:19:33.805 [debug] 'Connection closed', nil, nil
```

SSH SERVER WITH ELIXIR SHELL

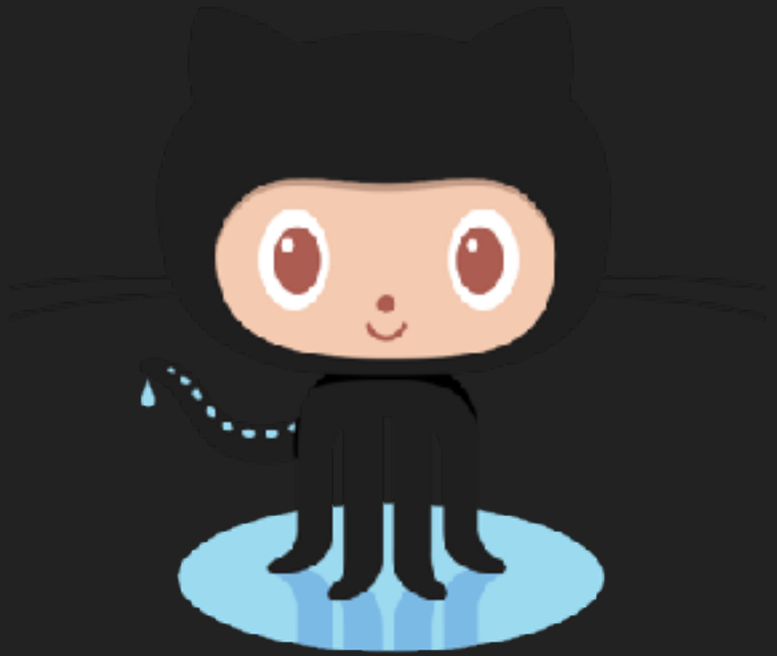


```elixir
def server_with_elixir_cli do
  :ssh.daemon 2225,
    system_dir: String.to_charlist(@root_dir),
    user_dir: String.to_charlist(@root_dir),
    auth_methods: 'publickey',
    disconnectfun: &log_it/1,
    connectfun: &log_it/3,
    failfun: &log_it/3,
    shell: &shell/2
end

def shell(username, peer) do
  Logger.debug("shell #{inspect username}, #{inspect peer}")
  IEx.start([])
end
```

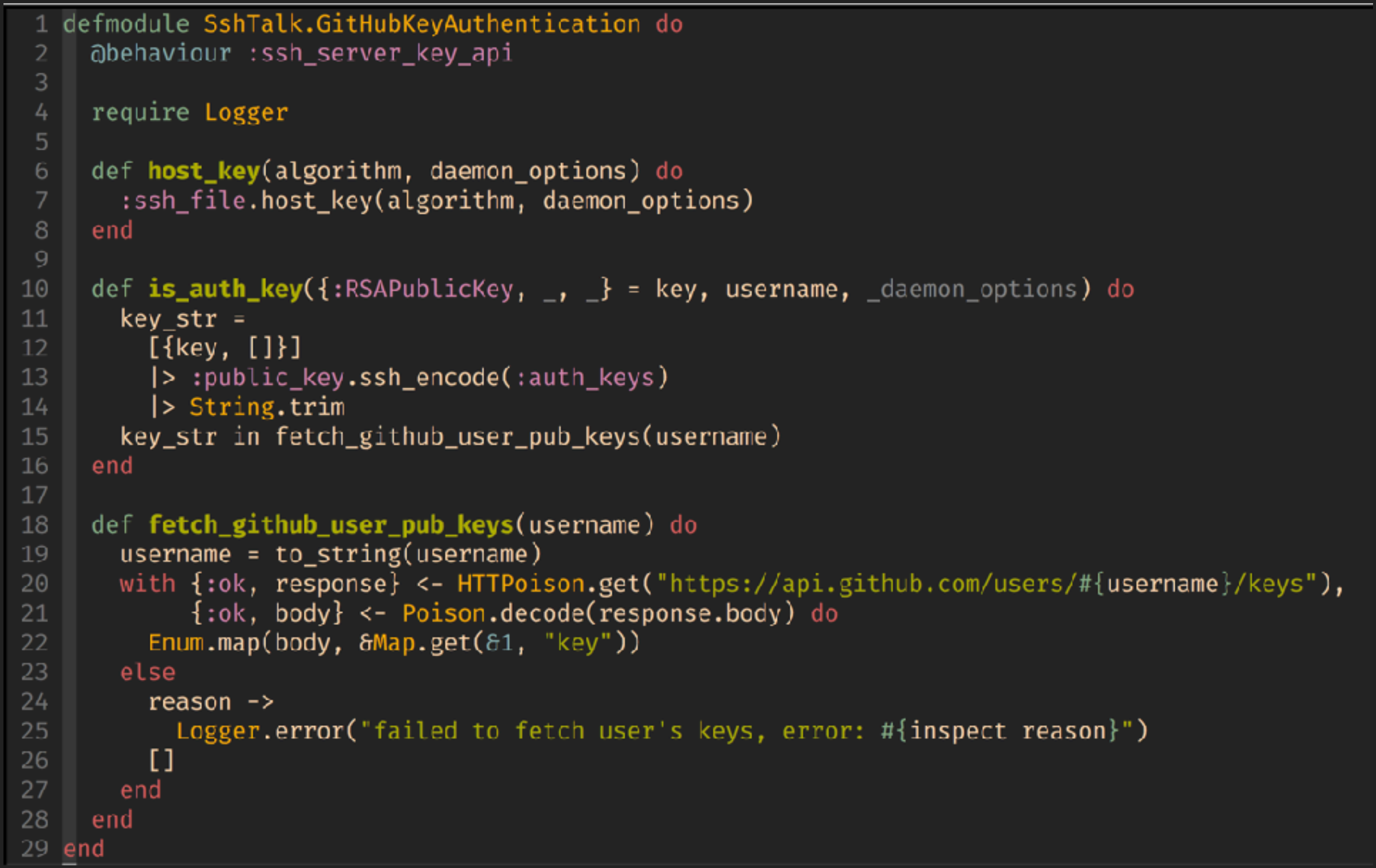


```
/p/t/ssh_talk $ ssh foo@localhost -p 2225
Interactive Elixir (1.5.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> SshTalk.hello
:world
iex(2)>
```

SSH SERVER AUTH PUBLIC KEY USING GITHUB ACCOUNT

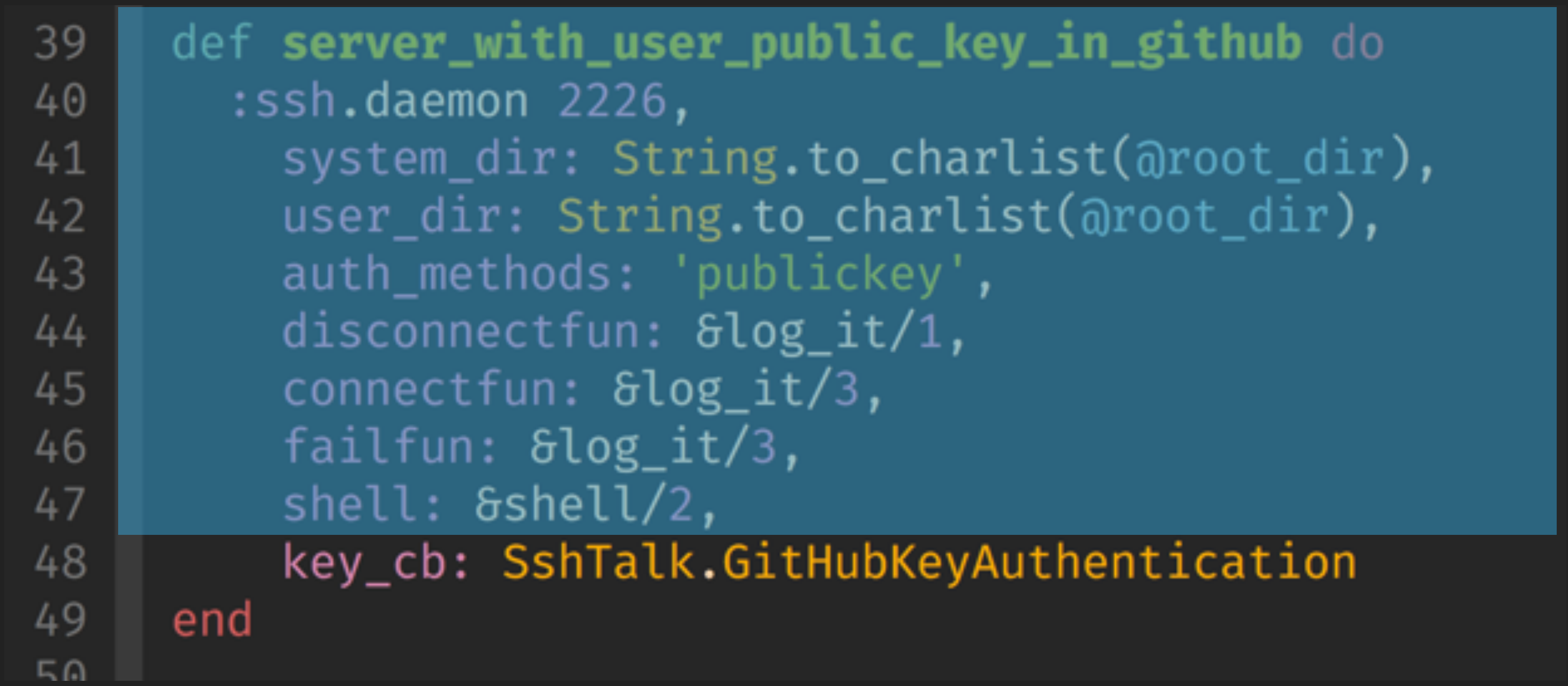


SSH SERVER AUTH PUBLIC KEY USING GITHUB ACCOUNT

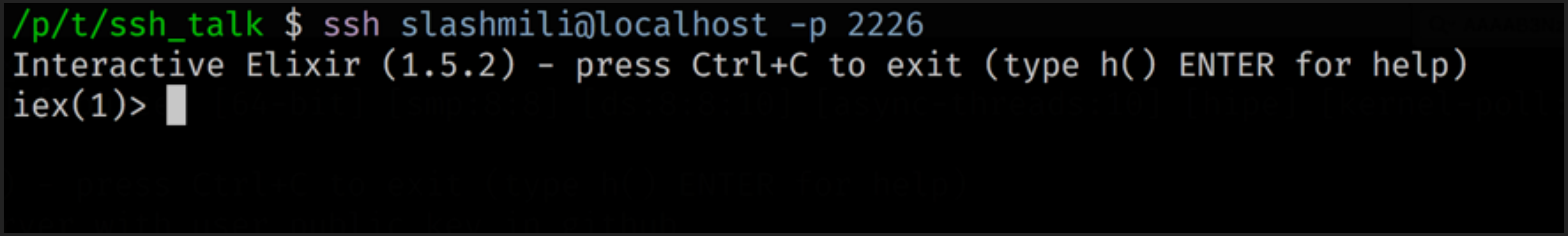


```elixir
defmodule SshTalk.GitHubKeyAuthentication do
  @behaviour :ssh_server_key_api

  require Logger

  def host_key(algorithm, daemon_options) do
    :ssh_file.host_key(algorithm, daemon_options)
  end

  def is_auth_key({:RSAPublicKey, _, _} = key, username, _daemon_options) do
    key_str =
      [{key, []}]
      |> :public_key.ssh_encode(:auth_keys)
      |> String.trim
    key_str in fetch_github_user_pub_keys(username)
  end

  def fetch_github_user_pub_keys(username) do
    username = to_string(username)
    with {:ok, response} <- HTTPoison.get("https://api.github.com/users/#{username}/keys"),
         {:ok, body} <- Poison.decode(response.body) do
      Enum.map(body, &Map.get(&1, "key"))
    else
      reason ->
        Logger.error("failed to fetch user's keys, error: #{inspect reason}")
        []
    end
  end
end
```

SSH SERVER AUTH PUBLIC KEY USING GITHUB ACCOUNT



```elixir
def server_with_user_public_key_in_github do
  :ssh.daemon 2226,
    system_dir: String.to_charlist(@root_dir),
    user_dir: String.to_charlist(@root_dir),
    auth_methods: 'publickey',
    disconnectfun: &log_it/1,
    connectfun: &log_it/3,
    failfun: &log_it/3,
    shell: &shell/2,
    key_cb: SshTalk.GitHubKeyAuthentication
end
```



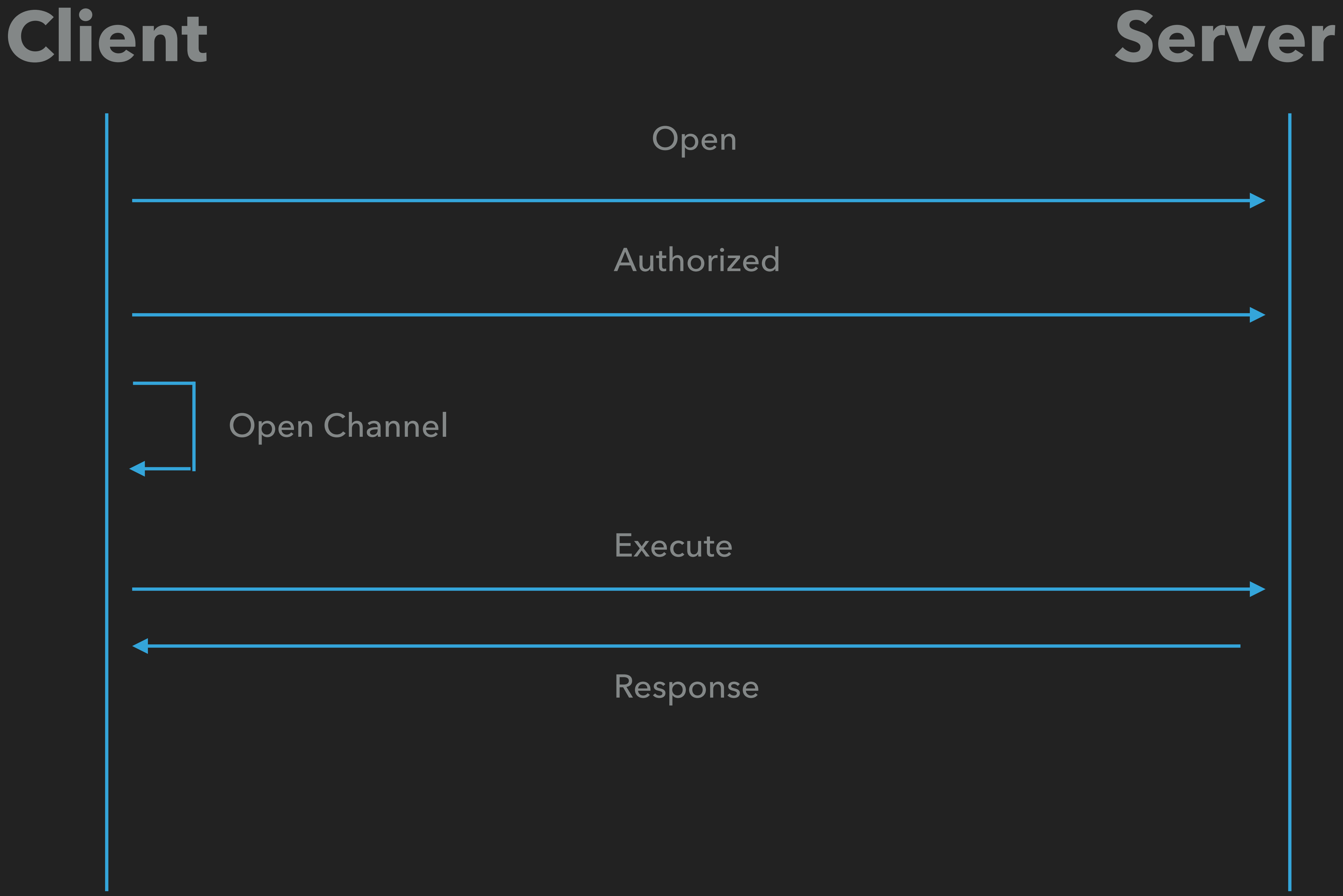
```
/p/t/ssh_talk $ ssh slashmili@localhost -p 2226
Interactive Elixir (1.5.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> █
```

SSH SERVER AUTH PUBLIC KEY USING GITHUB ACCOUNT

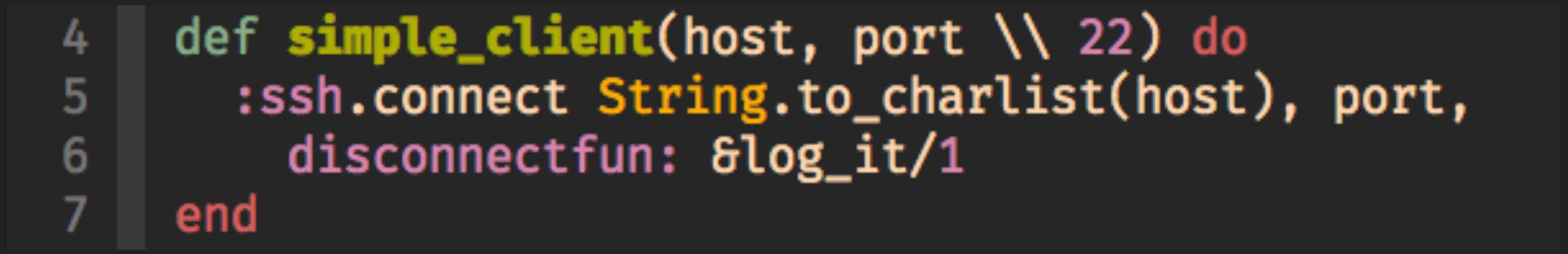
```
ssh <github-user>@ssh-talk.xyz
```

SSH CLIENT

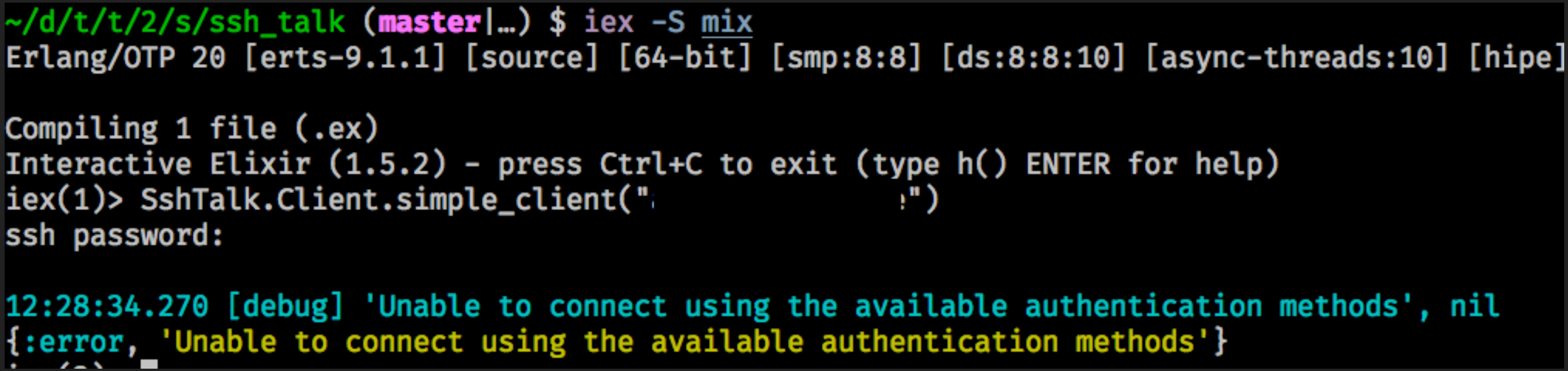
SSH CLIENT


Client
Server
Open
Authorized
Open Channel
Execute
Response

SSH CLIENT USING PASSWORD

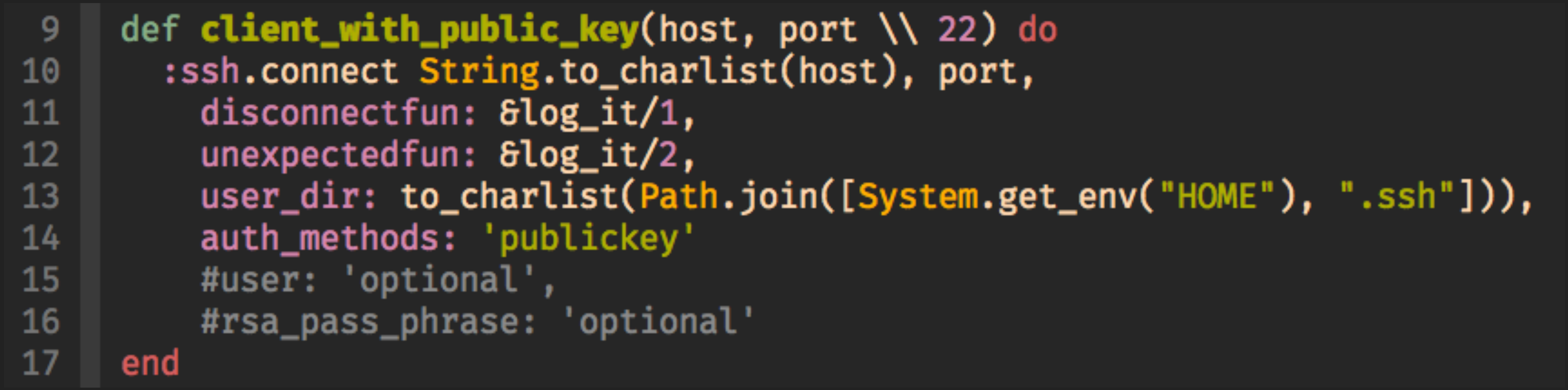


```elixir
def simple_client(host, port \\ 22) do
  :ssh.connect String.to_charlist(host), port,
    disconnectfun: &log_it/1
end
```

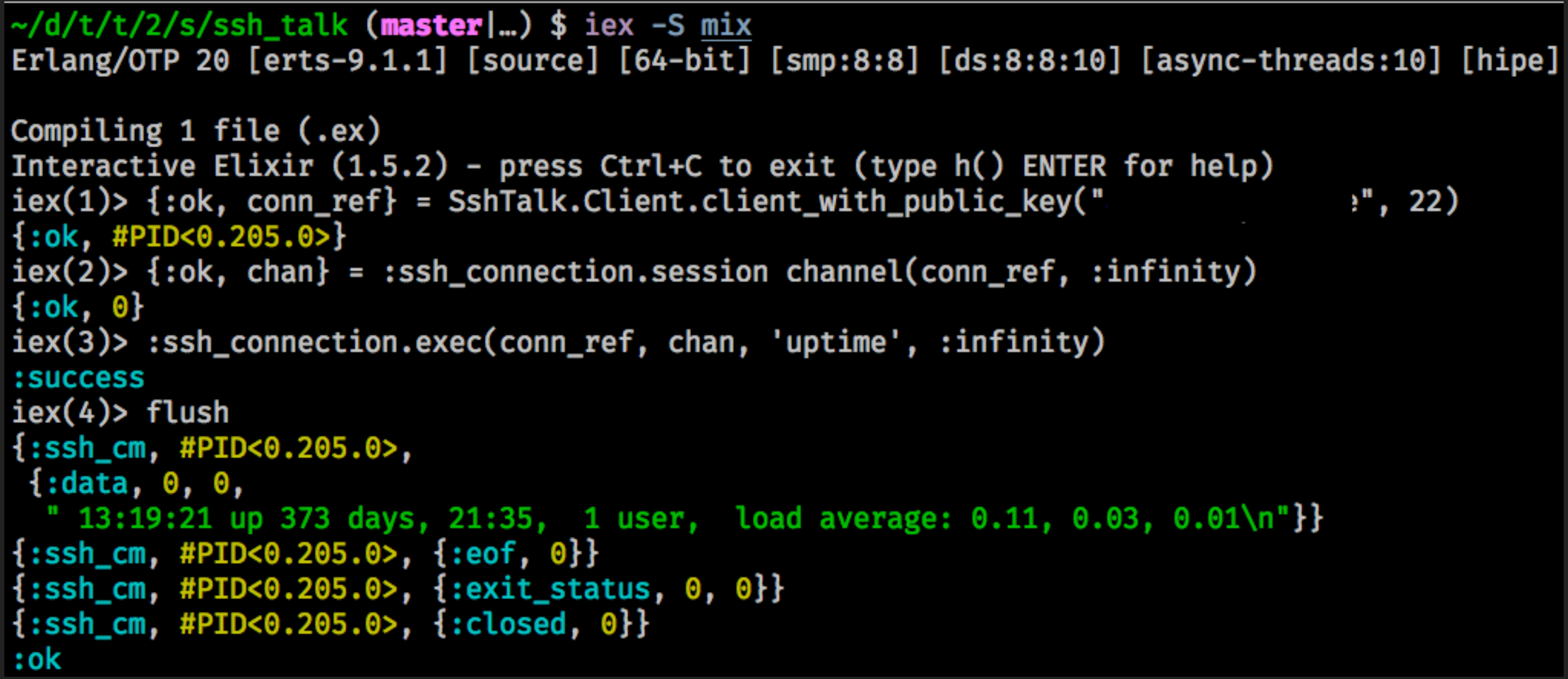


```
~/d/t/t/2/s/ssh_talk (master|…) $ iex -S mix
Erlang/OTP 20 [erts-9.1.1] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:10] [hipe]

Compiling 1 file (.ex)
Interactive Elixir (1.5.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> SshTalk.Client.simple_client("[redacted]")
ssh password:

12:28:34.270 [debug] 'Unable to connect using the available authentication methods', nil
{:error, 'Unable to connect using the available authentication methods'}
iex(2)>
```

SSH CLIENT AUTH WITH PUBLIC KEY



```elixir
def client_with_public_key(host, port \\ 22) do
  :ssh.connect String.to_charlist(host), port,
    disconnectfun: &log_it/1,
    unexpectedfun: &log_it/2,
    user_dir: to_charlist(Path.join([System.get_env("HOME"), ".ssh"])),
    auth_methods: 'publickey'
    #user: 'optional',
    #rsa_pass_phrase: 'optional'
end
```



```
~/d/t/t/2/s/ssh_talk (master|…) $ iex -S mix
Erlang/OTP 20 [erts-9.1.1] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:10] [hipe]

Compiling 1 file (.ex)
Interactive Elixir (1.5.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> {:ok, conn_ref} = SshTalk.Client.client_with_public_key("[redacted]", 22)
{:ok, #PID<0.205.0>}
iex(2)> {:ok, chan} = :ssh_connection.session_channel(conn_ref, :infinity)
{:ok, 0}
iex(3)> :ssh_connection.exec(conn_ref, chan, 'uptime', :infinity)
:success
iex(4)> flush
{:ssh_cm, #PID<0.205.0>,
 {:data, 0, 0,
  " 13:19:21 up 373 days, 21:35,  1 user,  load average: 0.11, 0.03, 0.01\n"}}
{:ssh_cm, #PID<0.205.0>, {:eof, 0}}
{:ssh_cm, #PID<0.205.0>, {:exit_status, 0, 0}}
{:ssh_cm, #PID<0.205.0>, {:closed, 0}}
:ok
```

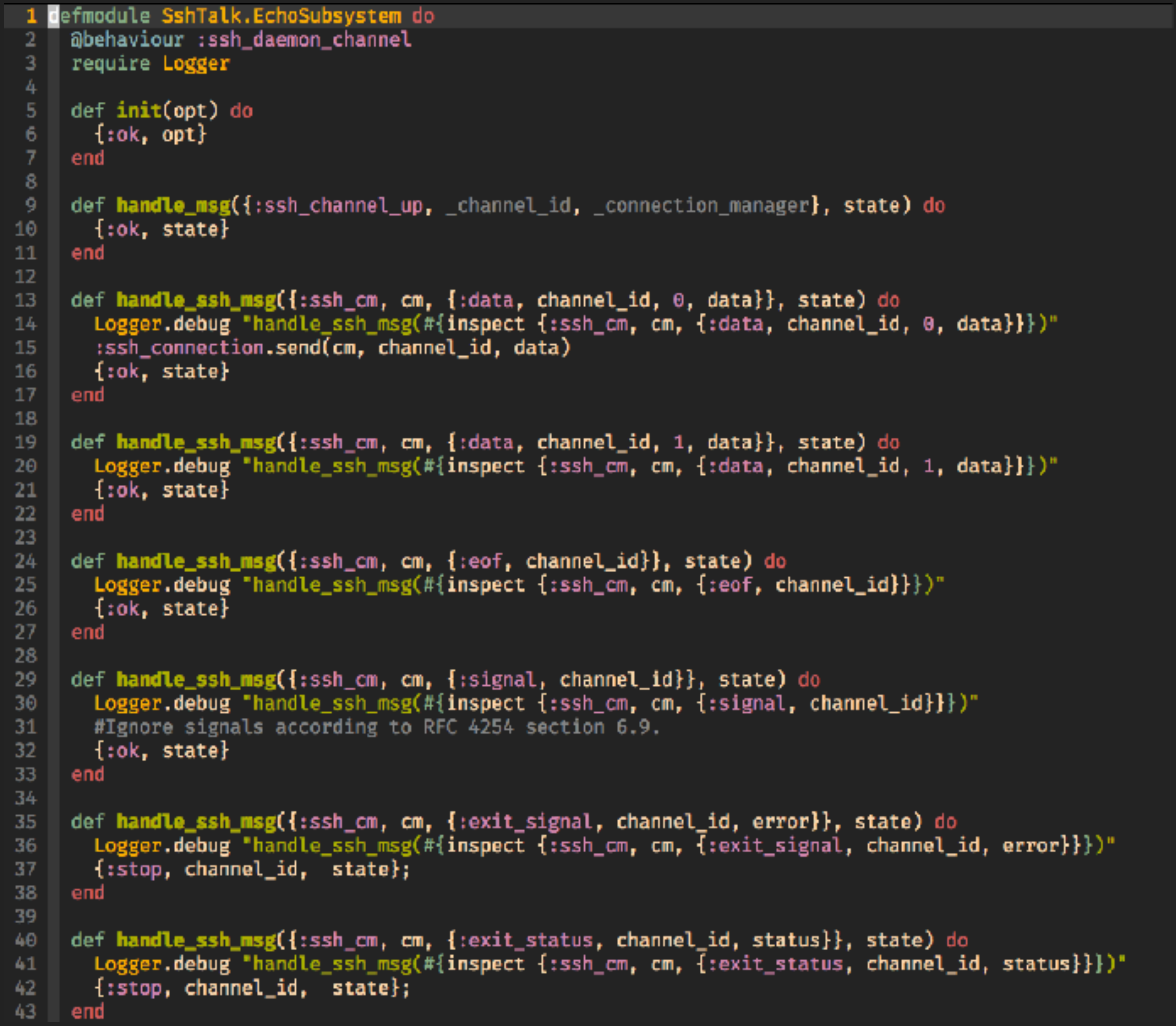
One more thing…

WOULDN'T IT BE NICE IF TWO ERLANG NODES COULD COMMUNICATE USING

SSH

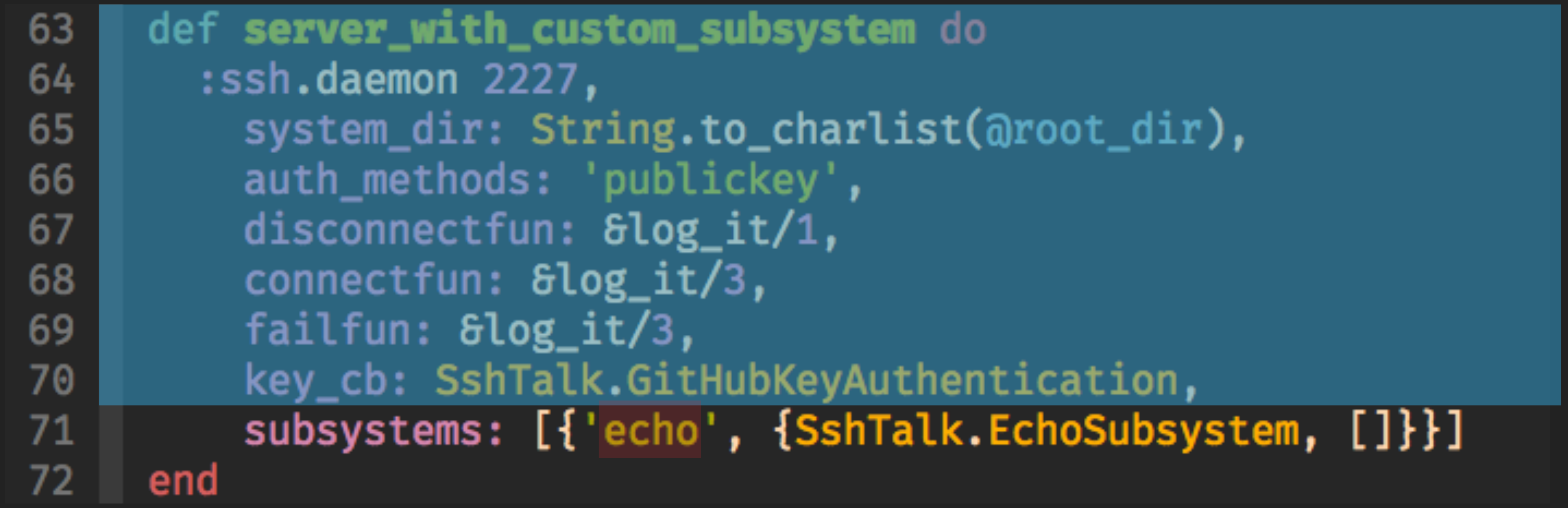
🤔

SSH SUBSYSTEM

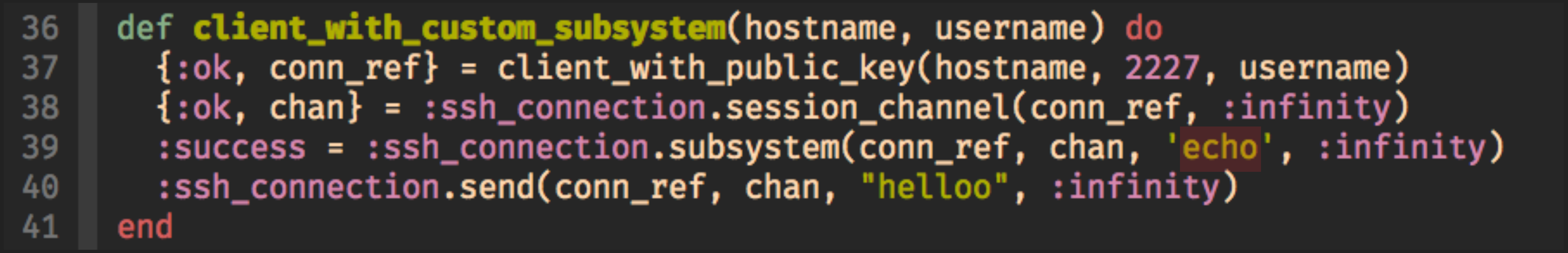


```
defmodule SshTalk.EchoSubsystem do
  @behaviour :ssh_daemon_channel
  require Logger

  def init(opt) do
    {:ok, opt}
  end

  def handle_msg({:ssh_channel_up, _channel_id, _connection_manager}, state) do
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:data, channel_id, 0, data}}, state) do
    Logger.debug "handle_ssh_msg(#{inspect {:ssh_cm, cm, {:data, channel_id, 0, data}}})"
    :ssh_connection.send(cm, channel_id, data)
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:data, channel_id, 1, data}}, state) do
    Logger.debug "handle_ssh_msg(#{inspect {:ssh_cm, cm, {:data, channel_id, 1, data}}})"
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:eof, channel_id}}, state) do
    Logger.debug "handle_ssh_msg(#{inspect {:ssh_cm, cm, {:eof, channel_id}}})"
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:signal, channel_id}}, state) do
    Logger.debug "handle_ssh_msg(#{inspect {:ssh_cm, cm, {:signal, channel_id}}})"
    #Ignore signals according to RFC 4254 section 6.9.
    {:ok, state}
  end

  def handle_ssh_msg({:ssh_cm, cm, {:exit_signal, channel_id, error}}, state) do
    Logger.debug "handle_ssh_msg(#{inspect {:ssh_cm, cm, {:exit_signal, channel_id, error}}})"
    {:stop, channel_id,  state};
  end

  def handle_ssh_msg({:ssh_cm, cm, {:exit_status, channel_id, status}}, state) do
    Logger.debug "handle_ssh_msg(#{inspect {:ssh_cm, cm, {:exit_status, channel_id, status}}})"
    {:stop, channel_id,  state};
  end
```

SSH SERVER WITH SUBSYSTEM

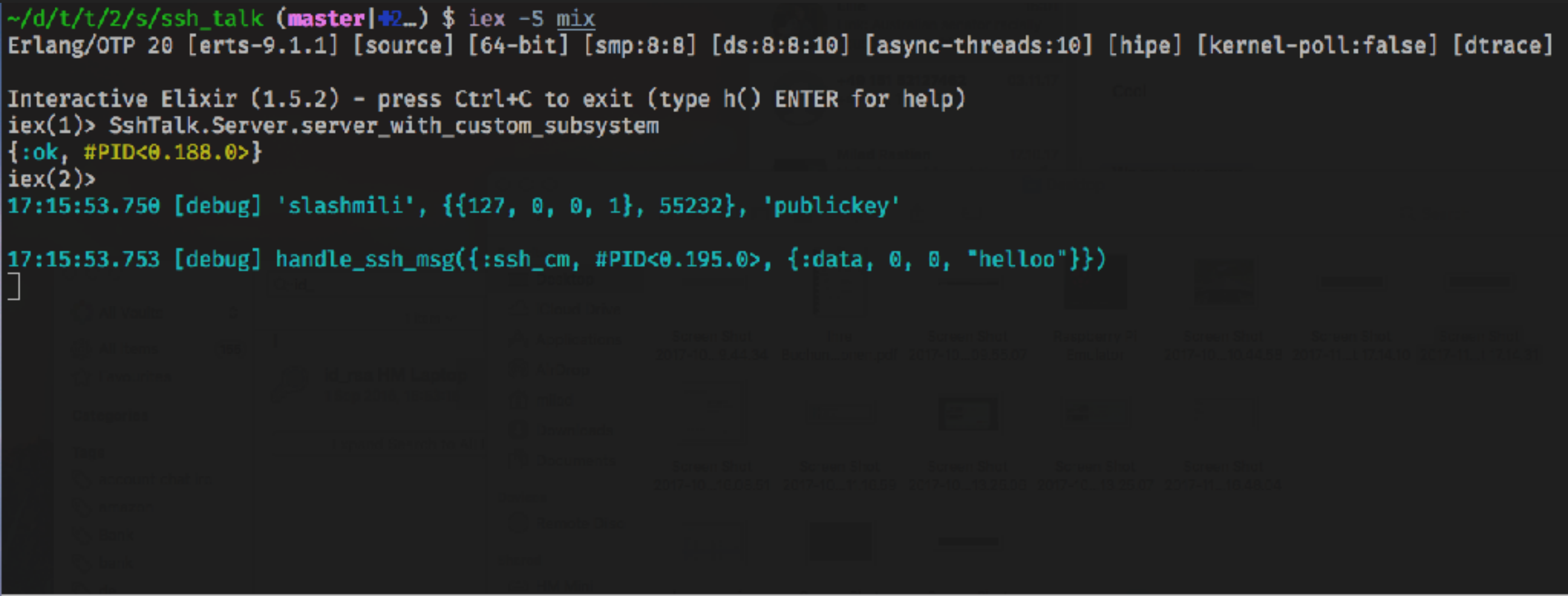


```
def server_with_custom_subsystem do
  :ssh.daemon 2227,
    system_dir: String.to_charlist(@root_dir),
    auth_methods: 'publickey',
    disconnectfun: &log_it/1,
    connectfun: &log_it/3,
    failfun: &log_it/3,
    key_cb: SshTalk.GitHubKeyAuthentication,
    subsystems: [{'echo', {SshTalk.EchoSubsystem, []}}]
end
```

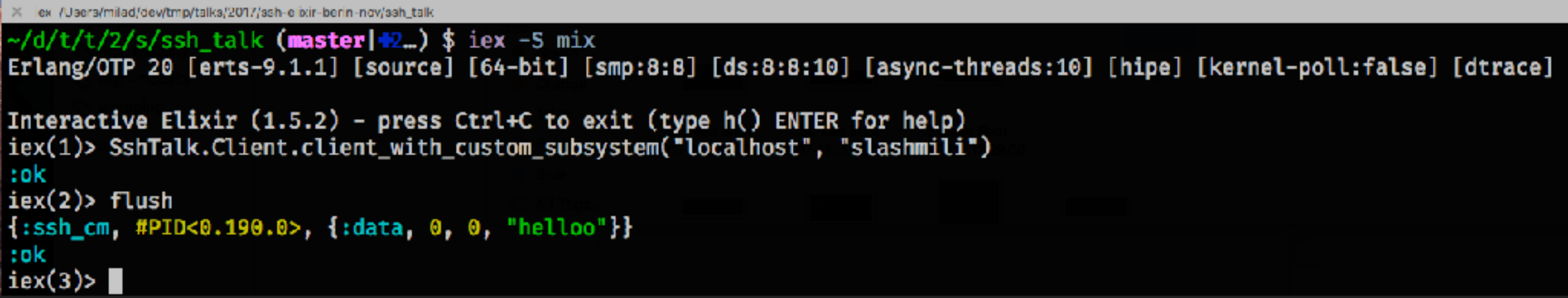


```
def client_with_custom_subsystem(hostname, username) do
  {:ok, conn_ref} = client_with_public_key(hostname, 2227, username)
  {:ok, chan} = :ssh_connection.session_channel(conn_ref, :infinity)
  :success = :ssh_connection.subsystem(conn_ref, chan, 'echo', :infinity)
  :ssh_connection.send(conn_ref, chan, "helloo", :infinity)
end
```

SSH SERVER WITH SUBSYSTEM EXAMPLE



```
~/d/t/t/2/s/ssh_talk (master|✚2…) $ iex -S mix
Erlang/OTP 20 [erts-9.1.1] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Interactive Elixir (1.5.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> SshTalk.Server.server_with_custom_subsystem
{:ok, #PID<0.188.0>}
iex(2)>
17:15:53.750 [debug] 'slashmili', {{127, 0, 0, 1}, 55232}, 'publickey'

17:15:53.753 [debug] handle_ssh_msg({:ssh_cm, #PID<0.195.0>, {:data, 0, 0, "helloo"}})
```



```
iex /Users/milad/dev/tmp/talks/2017/ssh-elixir-berlin-nov/ssh_talk
~/d/t/t/2/s/ssh_talk (master|✚2…) $ iex -S mix
Erlang/OTP 20 [erts-9.1.1] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Interactive Elixir (1.5.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> SshTalk.Client.client_with_custom_subsystem("localhost", "slashmili")
:ok
iex(2)> flush
{:ssh_cm, #PID<0.190.0>, {:data, 0, 0, "helloo"}}
:ok
iex(3)>
```

RESOURCES

- www.ssh-talk.xyz(until dec 2018!)
- http://erlang.org/doc/man/ssh.html
- http://erlang.org/doc/apps/ssh/using_ssh.html
- https://github.com/jbenden/esshd
- https://github.com/bitcrowd/sshkit.ex
- https://github.com/slashmili/talks/tree/master/2017/ssh-elixir-berin-nov

Q?

🙋‍♀️ 🙋‍♂️