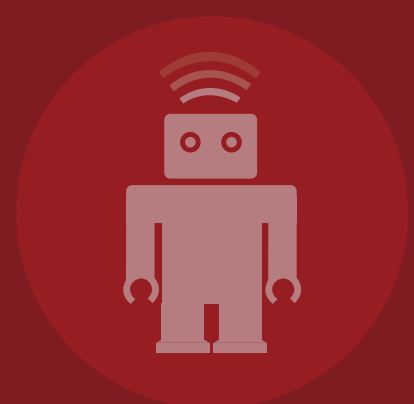
Angels & Daemons

Supporting your rails application with custom daemons

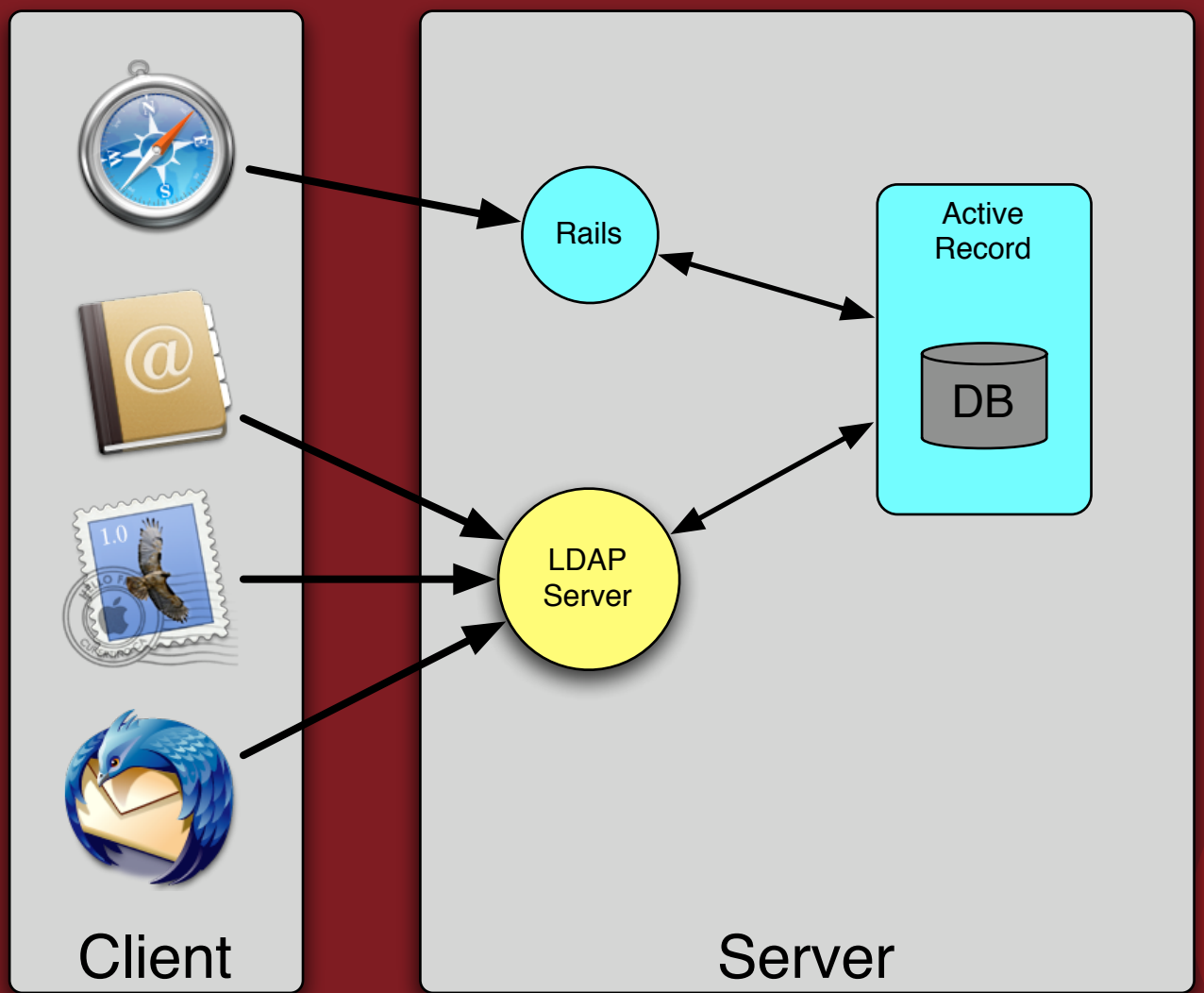
Tammer Saleh,
tsaleh@thoughtbot.com
Thoughtbot, Inc.

Our Problem

- Customer database in Rails
- Needed client-side integration -- mail, address book, outlook
- LDAP to ActiveRecord Gateway
  - Uses the ruby-ldapserver gem
  - Returns results from the DB



Our Solution

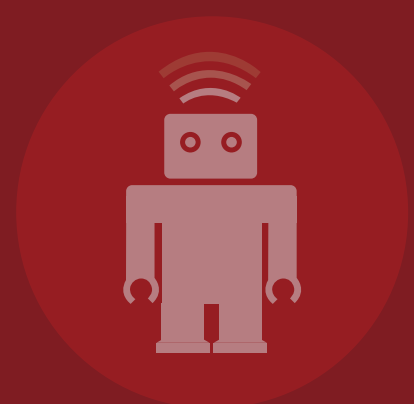


LDAP AR Gateway

http://thoughtbot.com/projects/ldap-ar-gateway

http://svn.thoughtbot.com/ldap-activerecord-gateway/

But how do we make it stick around?



NFSd

cups

Mongrel

Apache

Syslog

Cron

Samba

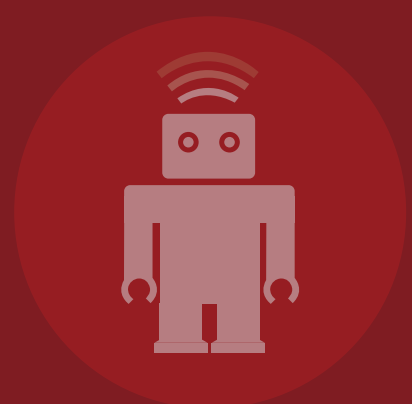
BackgroundDRb

sshd

MySQL

Memcached

FTPd



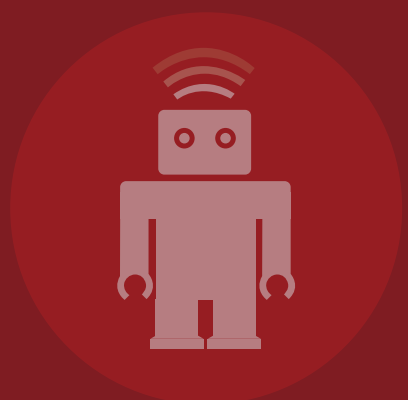
What's a Daemon?

Less than gods (OS & kernel), but more than mortals (programs)

From the Unix FAQ:

> A **daemon** process is usually defined as a background process that does not belong to a terminal session.

- No parent process
- No terminal
- Runs in background



Daemonizing

Fork off and die

```
def daemonize
  Kernel.fork and Kernel.exit









end
```

Daemonizing

Become a session and process group leader

```
def daemonize
  Kernel.fork and Kernel.exit
  Process.setsid




end
```

Daemonizing

Fork off and Die (again)

```
def daemonize
  Kernel.fork and Kernel.exit
  Process.setsid
  Kernel.fork and Kernel.exit




end
```

Daemonizing

Set umask

```
def daemonize
  Kernel.fork and Kernel.exit
  Process.setsid
  Kernel.fork and Kernel.exit
  File.umask 0




end
```

Daemonizing

Change to /

```
def daemonize
  Kernel.fork and Kernel.exit
  Process.setsid
  Kernel.fork and Kernel.exit
  File.umask 0
  Dir.chdir '/'



end
```

Daemonizing

Close inherited files

```
def daemonize
  Kernel.fork and Kernel.exit
  Process.setsid
  Kernel.fork and Kernel.exit
  File.umask 0
  Dir.chdir '/'
  ObjectSpace.each_object(IO) {|io| io.close rescue nil}



end
```

Daemonizing

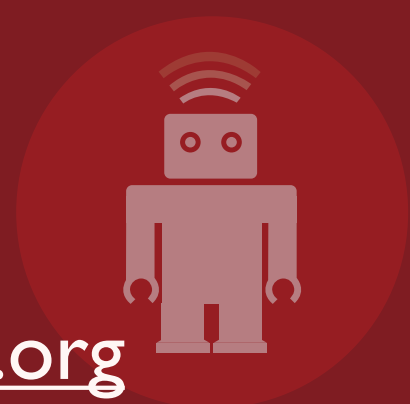
Reopen stdio

```
def daemonize
  Kernel.fork and Kernel.exit
  Process.setsid
  Kernel.fork and Kernel.exit
  File.umask 0
  Dir.chdir '/'
  ObjectSpace.each_object(IO) {|io| io.close rescue nil}
  STDIN.open( '/dev/null')
  STDOUT.open('/dev/null', 'a')
  STDERR.open('/dev/null', 'a')
end
```

Daemonizing

(the easy way)

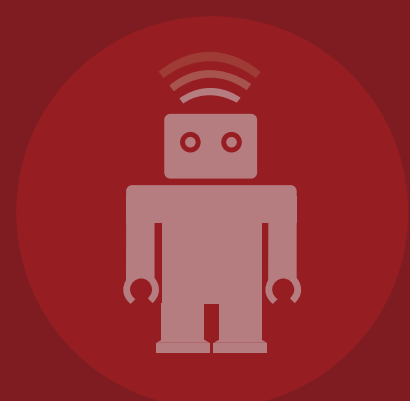
```
require 'daemons'
Daemons.daemonize

loop {
  # ..or whatever..
  conn = accept_conn()
  serve(conn)
}
```



Daemon gem: http://daemons.rubyforge.org

Great! Now what?

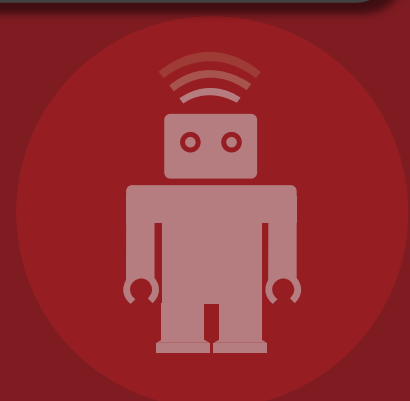
- Interacting with Rails
- Starting and stopping your daemon
- Ensuring only one is running at a time
- Config files
- Logging
- Security



Interacting with Rails

Load the Rails environment - DB, Models, Libraries, etc

```
require "/path/to/config/environment.rb"
if not defined? RAILS_ROOT
  raise RuntimeError, "Cannot load rails environment"
end
```

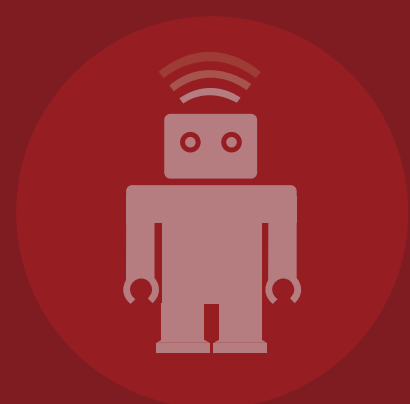


Interacting with Rails

Threads and Rails

```
ActiveRecord::Base.allow_concurrency = true
```

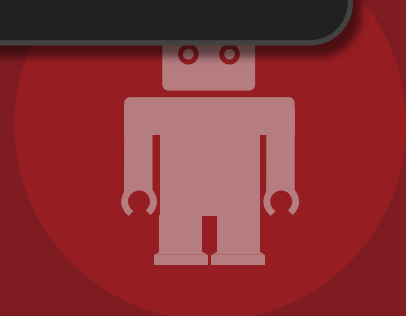
```
Mysql::Error: Lost connection to MySQL server during query:
```



Starting & Stopping

UNIX Init Scripts

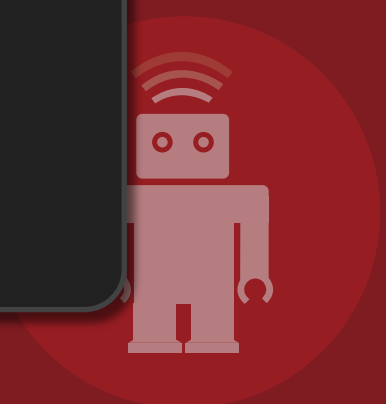
```ruby
#!/usr/bin/env ruby
basedir = File.expand_path(File.join(File.dirname(__FILE__), ".."))

require File.join(basedir, "lib", "server")

case ARGV[0]
  when "start":   Server.new(ARGV[1]).start
  when "stop":    Server.new(ARGV[1]).stop
  when "restart": Server.new(ARGV[1]).restart
  else puts "Usage: #{File.basename(__FILE__)} {start|stop|restart} arg"
end
```



Starting & Stopping

OS X Launchd

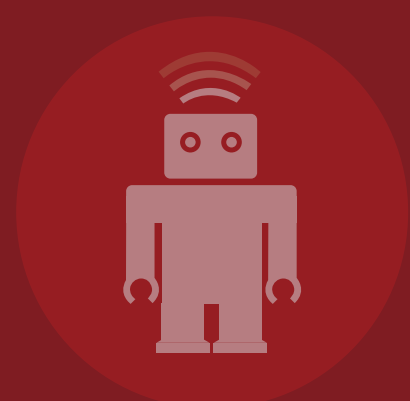
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>My Daemon</key>
        <string>localhost.etc.rc.command</string>
        <key>ProgramArguments</key>
        <array>
            <string>/path/to/daemon</string>
            <string>argument1</string>
            <string>argument2</string>
        </array>
        <key>RunAtLoad</key>
        <true/>
    </dict>
</plist>
```



http://developer.apple.com/macosx/launchd.html

Pid Files

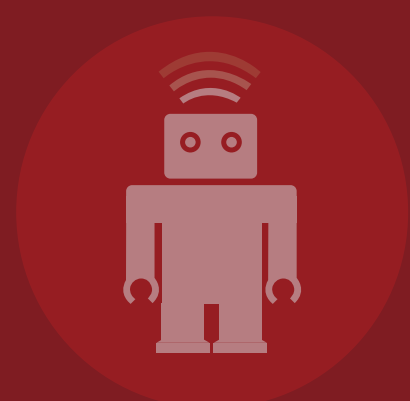
```
class PidFile
  def initialize(file)
    @file = file
  end

  def pid
    File.file?(@file) and IO.read(@file)
  end

  def remove
    if self.pid
      FileUtils.rm @file
    end
  end

  def create
    File.open(@file, "w") { |f| f.write($$) }
  end
end
```



Configuration Files

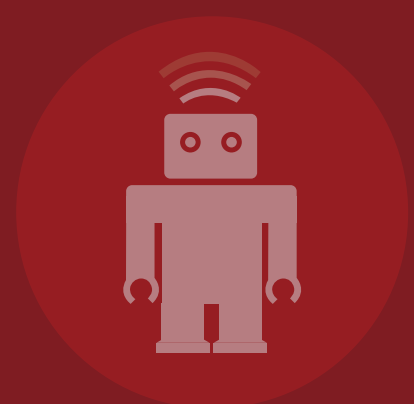
```
rails_dir: /path/to/rails/application/
debug: false # comment
complex:
  part1: <%= puts 'true' %>
  part2: false
```

```
@config =
  YAML.load(
    ERB.new(
      File.read("config_file")
    ).result
  ).symbolize_keys
```



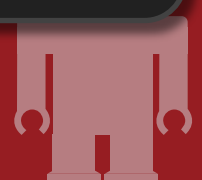
Logging

```
@logger = Logger.new("#{basedir}/log/server.log", 7, 1048576)

@logger.level = @config[:debug] ? Logger::DEBUG : Logger::INFO

@logger.datetime_format = "%H:%M:%S"
```



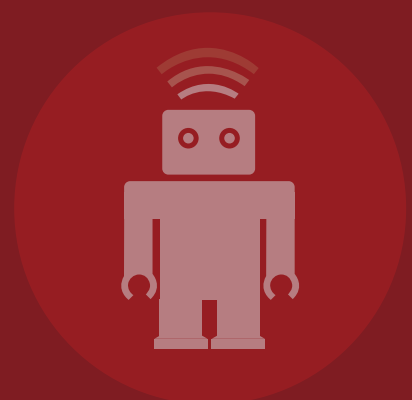
Dropping Privileges

```
def become_user(username = 'nobody', chroot = false)
  user = Etc::getpwnam(username)

  Dir.chroot(user.dir) and Dir.chdir('/') if chroot

  Process::initgroups(username, user.gid)
  Process::Sys::setegid(user.gid)
  Process::Sys::setgid(user.gid)
  Process::Sys::setuid(user.uid)
end
```



Testing

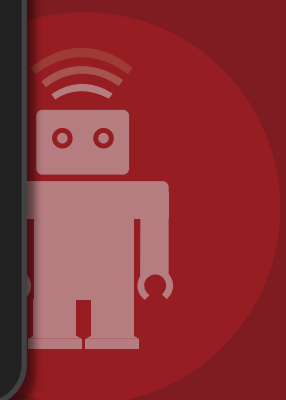
- Spawn daemon before tests
  - Complicated
  - Broken daemon may not die
  - Tests interfere with each other



Testing

Mocking makes this much safer

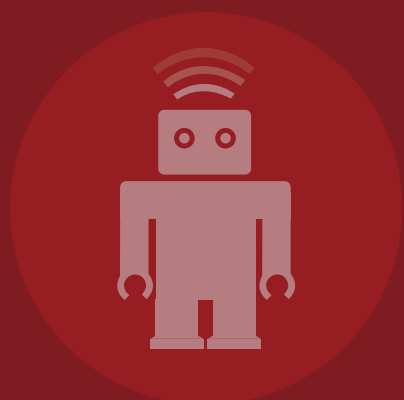
```
require "daemon"

class DaemonTest < Test::Unit::TestCase
  include Daemon
  def test_should_take_steps_to_daemonize
    Kernel.expects(:fork).times(2).returns(true)
    Kernel.expects(:exit).times(2)
    Process.expects(:setsid)
    File.expects(:umask).with(0)
    Dir.expects(:chdir).with('/')
    ObjectSpace.each_object(IO) { |io| io.expects(:close) }
    STDIN.expects( :reopen).with("/dev/null")
    STDOUT.expects(:reopen).with("/dev/null", "a")
    STDERR.expects(:reopen).with("/dev/null", "a")

    daemonize
  end
end
```



Alternatives

Rake tasks or ./script/runner

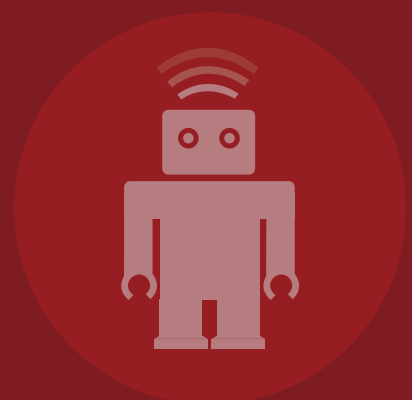
- Easy to write
- Launch via cron or by hand
- Only good for single-shot or periodic tasks
- Overlapping execution problem
- Crontabs can be a pain



Alternatives

nohup

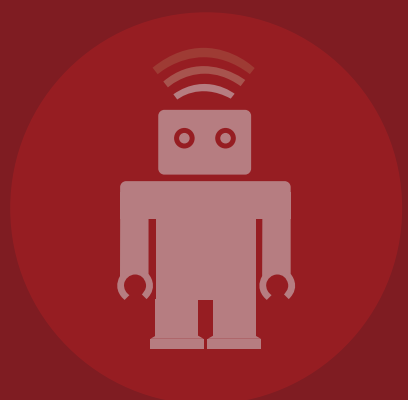
- Simple: just traps HUP signal
- Easy to use (“nohup script.rb”)
- Not as powerful or configurable



Alternatives

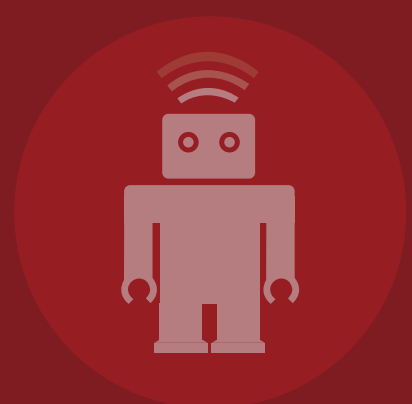
inetd & xinetd

- Easily write internet services
- Get tcp wrappers for free (usually)
- Not very efficient: must spawn application for each connection
- Not as portable



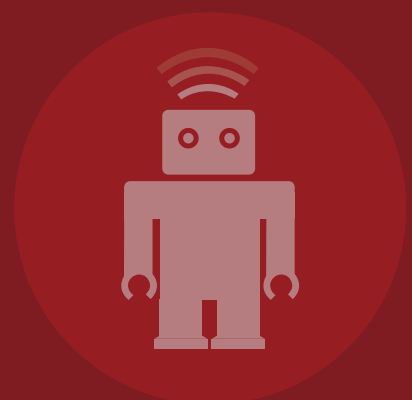
Final Tips

- Daemonize as soon in your code
- Rescue anything that could fail
- logger.debug with vigor



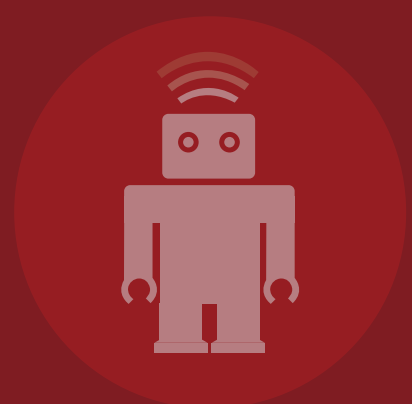
Where can we go from here?

- Other ways of interfacing with your data
  - WebDAV
  - SNMP
  - FTP
- System monitoring
- Long-running or CPU intensive tasks



Where can we go from here?

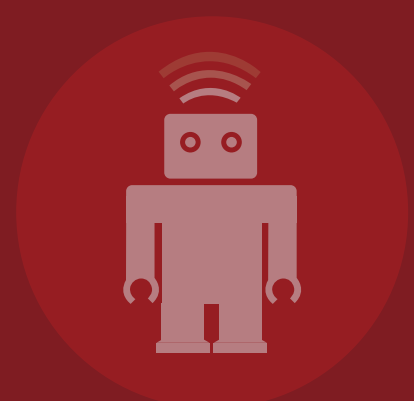
- Interfacing with other apps through REST
  - FTP ↔ Highrise
  - IRC ↔ Campfire proxy



Where can we go from here?

Highrise to LDAP proxy

http://svn.thoughtbot.com/highrise-ldap-proxy



Thanks to

Thoughtbot.com

Brian Candler

for the ruby-ldapserver gem

http://raa.ruby-lang.org/project/ruby-ldapserver/