# Episode 10: Mix

## Summary

- Mix intro
- Demo: Mix Project, Mix Tasks

## Mix

Mix is a Build tool for Elixir

- Project organization
- Dependency management
- Build tasks
- Custom tasks

### Commands

| Command | Description |
|---|---|
| `mix new project_name` | New mix project |
| `mix new project_name --umbrella` | New umbrella project |
| `cd project_name/apps && mix new component_name` | Create component project within umbrella project |

## Choosing an environment

In the terminal, run `MIX_ENV=<environment>`, where `<environment>` is one of:

- `prod`
- `test`
- `dev`

Examples

```
MIX_ENV=dev iex -S mix
MIX_ENV=prod mix compile
```

> See the section *Environment configuration* below to learn how to configure each environment separately.

## Running `iex`

- To run `iex` in the context of a Mix project, run `iex -S mix`
- In `iex`, run `recompile/1` to recompile the project

> – **Note**: The application is not restarted after compilation, which means any long running process may crash as any changed module will be temporarily removed and recompiled, without going through the proper code changes callback. (src)

## Parts of a Project

Part | Description
`mix.exs` | Project name, version, dependencies
`/test` | ExUnit tests
`/lib` | Source files
`/config/config.exs` | Project configuration

### Organizing source files

```
root
/lib
  project_name.ex
  /project_name
```

- `/lib/project_name.ex` often used for high-level documentation
- All source files go in `/lib/project_name`.
- Source files should be named as submodules of `ProjectName`

## Project configuration

- In `mix.exs`, `config/2` configures the given application
- Project options can be accessed via `Application.get_env/2`
  - Tip: use module attributes (`@name value`) to reduce verbosity

### Configuring the entire project

```
# mix.exs: Options for entire project
config :project_name, some_option: value, ...

# In an app: Retrieving options
Application.get_env(:project_name, :some_option)
# => value
```

### Configuring a particular app

```
# mix.exs: Options for some app at /lib/ProjectName/MyApp.ex
config :project_name, ProjectName.MyApp, some_option: value, ...

# MyApp.ex: Retrieving options (2 ways)
Application.get_env(:project_name, __MODULE__) # or,
Application.get_env(:project_name, ProjectName.MyApp)
# => [some_option: value]
```

### Environment configuration (`prod`, `test` and `dev`)

1. Uncomment `import_config "#{Mix.env}.exs"` in `mix.exs`
2. Create [`prod`|`test`|`dev`]`.exs` files in `/config`, depending on which ones you need
3. Type `use Mix.Config` in each `<environment>.exs` file

## Tasks

Here are a few notable tasks.

| | Description |
|---|---|
| `mix help` | List available tasks |
| `mix compile` | Compiles project for current environment |
| `mix test` | Runs tests in `/test` |

**Writing your own tasks**

1. Create directories: `mkdir -p /lib/mix/tasks`
2. Create file: `touch /lib/mix/tasks/<task>.ex`
3. Create module: `Mix.Tasks.<Task>`; see sample below

```
defmodule Mix.Tasks.Encrypt do
  @shortdoc "Encrypts some arbitrary text"

  @moduledoc """
  Encrypts and prints text from the given text option.

      mix encrypt -t "Something fun!"

  ## Command line options

  - -t, --text - text to encrypt and print
  """

  use Mix.Task

  @doc """
  Encrypts given text and prints to console.

  ## Examples

      $ MIX_ENV=prod
      $
  """
  @spec run(args :: keyword(String.t)) :: :ok | :error
  def run(nil), do: IO.puts "Run mix help encrypt for command line
  ↪  options."
  def run(args) do
    {opts, _, _} = OptionParser.parse(args, aliases: [t: :text])
    IO.puts Crypto.Encryptor.encrypt(opts[:text])
  end
end
```