mods: community modifications

mods are small chunks of code to create custom modifications to the core workings of the norns system software. mods are features which modify the basic functionality of norns, for all scripts, but is a feature which isn't necessary to include in the foundational norns codebase.

installing a mod

mods can be installed through the maiden project manager, alongside other community scripts. Once you've installed a mod, be sure to restart norns using `SYSTEM > RESTART` and move to the instructions in the next section.

enabling / disabling a mod

By their very nature, mods can have a negative impact on system stability or make system level changes which may not be universally welcome. Even if installed, a mod will *by default* start as unloaded. One must explicitly enable a mod and restart norns before its functionality will be available.

To enable a mod, visit `SYSTEM > MODS` to see the list of installed mods. Mods which are loaded will have a small dot to the left of their name. Use E2 to selected an item in the list and then use E3 to enable or disable as appropriate. Unloaded mods will show a `+` to the right their name to indicate that they will be enabled (and thus loaded) on restart. Loaded mods will show a `-` to the right of their name indicating they will be disabled on restart.

Loaded mods which have a menu will have `>` at the end of their name. Pressing K3 will enter the mod's menu.

writing a mod

A mod lives in the `dust` folder just like standard norns scripts. Unlike norns scripts, the code in a mod is loaded by matron when it starts up. Mods can modify or extend the Lua environment globally such that the changes are visible to all scripts.

mods work by registering functions to be called by matron hooks. matron defines five hooks:

- `system_post_startup` - called after matron has fully started and system state has been restored but before any script is run
- `system_pre_shutdown` - called when `SYSTEM > SLEEP` is selected from the menu
- `script_pre_init` - called after a script has been loaded but before its engine has started, pmap settings restored, and `init()` function called
- `script_post_init` - called after a script's `init()` function has been called
- `script_post_cleanup` - called after a script's `cleanup()` function has been called, this normally occurs when switching between scripts

Multiple mods can each register their own callback function for a given hook. matron will sort the hooks alphabetically before executing them. This allows mods that add parameters to add them in a consistent order – eg. mods that need all parameters available before executing should start with `~` to ensure they're at the end.

Any errors generated within a registered function will be caught and ignored in the maiden REPL.

mods may additionally provide their own menu page accessible via the `SYSTEM > MODS` menu.

example mod

A mod is any norns script/project which contains a file called `lib/mod.lua`. This layout allows a mod to be either a standalone project or simply functionality bundled along side an existing norns script. A mod is free to `require` additional modules as needed.

This example demonstrates the mod development approach:

```
--
-- require the `mods` module to gain access to hooks, menu, and other utility
-- functions.
--

local mod = require 'core/mods'

--
-- [optional] a mod is like any normal lua module. local variables can be used
-- to hold any state which needs to be accessible across hooks, the menu, and
-- any api provided by the mod itself.
--
-- here a single table is used to hold some x/y values
--
```

```
local state = {
  x = 0,
  y = 0,
}



--
-- [optional] hooks are essentially callbacks which can be used by multiple mods
-- at the same time. each function registered with a hook must also include a
-- name. registering a new function with the name of an existing function will
-- replace the existing function. using descriptive names (which include the
-- name of the mod itself) can help debugging because the name of a callback
-- function will be printed out by matron (making it visible in maiden) before
-- the callback function is called.
--
-- here we have dummy functionality to help confirm things are getting called
-- and test out access to mod level state via mod supplied fuctions.
--

mod.hook.register("system_post_startup", "my startup hacks", function()
  state.system_post_startup = true
end)

mod.hook.register("script_pre_init", "my init hacks", function()
  -- tweak global environment here ahead of the script `init()` function being called
end)



--
-- [optional] menu: extending the menu system is done by creating a table with
-- all the required menu functions defined.
--

local m = {}

m.key = function(n, z)
  if n == 2 and z == 1 then
    -- return to the mod selection menu
    mod.menu.exit()
  end
end

m.enc = function(n, d)
  if n == 2 then state.x = state.x + d
  elseif n == 3 then state.y = state.y + d end
  -- tell the menu system to redraw, which in turn calls the mod's menu redraw
  -- function
  mod.menu.redraw()
end

m.redraw = function()
  screen.clear()
  screen.move(64,40)
  screen.text_center(state.x .. "/" .. state.y)
  screen.update()
end

m.init = function() end -- on menu entry, ie, if you wanted to start timers
m.deinit = function() end -- on menu exit

-- register the mod menu
--
-- NOTE: `mod.this_name` is a convienence variable which will be set to the name
-- of the mod which is being loaded. in order for the menu to work it must be
```

```
-- registered with a name which matches the name of the mod in the dust folder.
--
mod.menu.register(mod.this_name, m)



--
-- [optional] returning a value from the module allows the mod to provide
-- library functionality to scripts via the normal lua `require` function.
--
-- NOTE: it is important for scripts to use `require` to load mod functionality
-- instead of the norns specific `include` function. using `require` ensures
-- that only one copy of the mod is loaded. if a script were to use `include`
-- new copies of the menu, hook functions, and state would be loaded replacing
-- the previous registered functions/menu each time a script was run.
--
-- here we provide a single function which allows a script to get the mod's
-- state table. using this in a script would look like:
--
-- local mod = require 'name_of_mod/lib/mod'
-- local the_state = mod.get_state()
--
local api = {}

api.get_state = function()
  return state
end

return api
```

---

help