they don't alter build process

Unofficial, opinionated NuttX project management tool

python scripts
Goals

– Simplify NuttX setup tasks
– Lead the user by suggesting next steps
– Change folder structure to keep project-dedicated code together
– Group configuration options into project-related, logical modules
Short command list

$ upm new [project_name]
$ upm generate
    --app [name]
    --app-template [example]
$ upm generate
    --lib [name]
    --lib-template [example]
$ upm config
$ upm run
$ upm clean
$ upm console
$ upm console --telnet

No need to be in nuttx directory.
Desired project structure

/apps – just custom apps
/libs – just custom libs
/boards – just custom board code
/os – upstream bundle

/nuttx
/apps (custom_apps → ../ ../ ../ apps, custom_libs → ../ ../ ../ libs)
Why do we need CLI tool? Usual tasks.

– How to clone repositories and build the firmware?
– How to add a new app?
– How to add a custom board?
– How to add some library?
– How to use Git for new app, custom board and custom library?
How to clone, build, flash?

$ mkdir firmware
$ cd firmware
$ git clone https://bitbucket.org/nuttx/nuttx.git nuttx
$ git clone https://bitbucket.org/nuttx/apps.git apps
$ cd nuttx
$ tools/configure.sh hymini-stm32v/usbnsh
$ make
$ st-flash write nuttx.bin 0x8000000
How to add a new app?

$ cd firmware
$ cd apps/examples
$ cp -r hello myapp
$ cd ../../nuttx
$ make

$ cd ../../../apps/examples/myapp
$ find . -type f -exec sed -i "s/hello/myapp/g" {} \;
$ find . -type f -exec sed -i "s/HELLO/mypapp/g" {} \;
$ mv hello_main.c myapp.cxx
$ make

crash during linking because of duplicate names
How to add a custom board code?

```bash
$ cd nuttx/configs
$ cp -r stm32f4discovery custom_board
```

```bash
nuttx/configs/Kconfig
...
+ config ARCH_BOARD_CUSTOM_BOARD
+   bool "Custom board"
+   depends on ARCH_CHIP_STM32F407VG || ARCH_CHIP_STM32F407ZG
...
+ default "custom_board" if ARCH_BOARD_CUSTOM_BOARD
...
+ if ARCH_BOARD_CUSTOM_BOARD
+   source "configs/custom_board/Kconfig"
+ endif
...
```
Why do we need CLI tool?

NuttX currently relies on simplicity
- Make, nuttx/tools
- Kconfig-frontends
- C/C++ preprocessor
- A few symlinks

Tool is not necessary
+ Simplicity
+ Full control over the process

But we could use one
+ Provides some guidance
+ Easier entry for new people
+ What is logically one thing should be one command
What CLI tools do in frameworks?

– Simplification of common tasks, like
  – starting new projects
  – building
  – versioning and distribution
  – handling updates
  – deploying

– Generation of boilerplate code
– Warning about issues, suggesting solutions

– They include some know-how which user don't need to remember
Example upm usage

fw $ upm init .
fw $ upm generate --app rest_api --app-template webserver
fw $ upm use --board stm32f4discovery
fw $ upm run
fw $ upm console
(enter)
nsh>

- Generates directory structure
- Clones NuttX repositories
- Copies selected example app
- Links selected board
- Builds and flashes, connects to NuttX console
Configuration protection

fw $ upm config

upm: do you accept the changes (y, n, q)? – rollback otherwise
Configuration modules – Kconfiglib

config/atoms/enc28j60.kconfig
NET=y
NET_TCP=y
NET_DEVICES=y
ENC28J60=y
ENC28J60_LPWORK=y
ENC28J60_HALFDUPLEX=y
NET_HOSTNAME=nuttx
NET_ROUTE=y
    # In order to reach destinations through default gateway
NET_ARP_SEND=y  # For instant connection

config/modules/webserver
enc28j60
http

$ upm config --batch webserver
Philosophy

– Do not perform actions, just suggest them, be verbose

$ upm init firmware
upm: created .nuttx, active, apps, boards, dist, libs, oses, tmp
upm: as a next step, you might wish to download upstream NuttX

    cd firmware
    upm clone --os default

– Stay simple
– Always ask before doing intrusive change
– Accept commands from any subdirectory (similar to git)
– Show state when asked (similar to git)
– Tries to get a bit bigger picture than scripts in nuttx/tools
How upm could handle Spresense™?

– suggest cloning of **SDK** and cloning it for user
– suggest downloading of **firmware, flash_writer**
  – ask user to accept licenses and download files to some folder
  – say where to find them
  – warn if something is still missing
  – etc.
    – warn if `python3-serial` module is missing

Maybe some plugin approach for platforms with extra requirements.
Challenges?

– some part still work in progress
– maybe disabling what is not polished would be a way to go
– dependency management?
– plugins
– git submodule management
REST APIs

Express-like + UI synchronization
## REST APIs – Node.js Express

```javascript
let express = require('express')
let mem = require('mem')
let app = express()

let memory_route = function (req, res) {
  res.json({
    total: mem.total,
    used: mem.used,
    largest: mem.largest
  })
}

app.get('/memory', memory_route)

app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
});
```
static void memory_route(struct httpd_state *pstate, char *ptr) {
    static const int memory_answer_max_size = 113;  // content length with all INT_MIN, +1
    char body[memory_answer_max_size];
    int body_len;

    body_len = snprintf(body, memory_answer_max_size, "\n"{%"memory": {\n    "total": %d,\n    "used": %d,\n    "largest": %d\n}}\n",
    mem.arena, mem_used.get(), mem_largest.get()
    );

    send_reply(pstate, 200, body, body_len);
}
REST APIs – Express-like list of routes

```c
static struct httpd_cgi_call routes[] = {
    { NULL, "/" , catchall_route },
    { NULL, "/os/memory" , memory_route },
    { NULL, "/os/time" , time_route },
    { NULL, "/console" , console_route },
    { NULL, "/sensors" , sensors_route },
    { NULL, "/adc" , adc_route },
    { NULL, "/state" , state_route }
};

http_server(routes, (sizeof routes / sizeof *routes));
```

Simple helpers:
- [https://gitlab.com/w8jcik/upm/snippets/1876008](https://gitlab.com/w8jcik/upm/snippets/1876008) (http.cxx)
- [https://gitlab.com/w8jcik/upm/snippets/1876010](https://gitlab.com/w8jcik/upm/snippets/1876010) (http.h)
Reaching μc from JavaScript framework

React, Vue, Ember
What with two windows open?
Reporting only changed state

... for (i = 0; i < reported_count; i += 1) {
    if (same_boot && previous_update_time >= (*all_reported[i]).get_changed_time()) {
        continue;
    }
}

any_reported = true;

body_len += snprintf(&body[body_len], answer_max_size, "\n%"s": %d,",
    (*all_reported[i]).get_name(),
    (*all_reported[i]).get()
);

... Variable.get_changed_time()
Variable.get_name()
Variable.get()

A lot of variables.
Two different browser windows synchronize
More efficient than polling

NuttX 2019
experience exchange :)
LittlevGL

Small UI apps and emulation
LittlevGL 6.0 released this week

Open-source Embedded GUI Library

Screen rotation
Multiple displays
LittlevGL 5.3

– Present in apps/examples
– Example board configuration for /dev/fb0
– But can be also used without framebuffer
  – External RAM not required
  – Display connected in any way (SPI, FSMC)
Running in emulator – Mocks

Makefile
CFLAGS += -DINSIDE_NUTTX=1
CXXFLAGS += -DINSIDE_NUTTX=1

static int adc_read(adc_msg_t *sample) {
    #ifdef INSIDE_NUTTX
        ...
    #endif
}

#ifdef INSIDE_NUTTX
    # include <graphics/lvgl.h>
#else
    # include "lvgl/lvgl.h"
#endif

#ifdef INSIDE_NUTTX
    # include <sys/ioctl.h>
    # include <nuttx/ioexpander/gpio.h>
    # include <sys/boardctl.h>
    # include "watchdog/watchdog.h"
#endif
Emulator
Microapps

Letting projects grow

Monolythic lvgl app
(not easily reusable)

Slight analogy to microservices and android intentions

lvgl initialization

Business logic
(not reusable)

lvgl deinitialization

Network configuration app

Time/date configuration app

Assets copying app

Slight analogy to microservices and android intentions
Development UI development stages

UI prototyping tool → InVision → LittlevGL Emulator → Microcontroller NuttX

(discussion and review)

(visual prototype)

Button
Challenges?

– integration of LittlevGL emulator with upm

  /oses
  /default (/nuttx, /apps)
  /lvgl_emulator

– lack of libraries for UI prototyping software
– check the process