

MCUSH 开发测试环境说明

说明：此文档描述如何搭建 MCUSH 开发测试环境，包括固件开发、串口调试和自动化测试工具的开发。

作者：彭树林

更新日期：2019-9-5

1、固件开发环境

操作系统

MCUSH 最早是在 Ubuntu Linux 环境下开发的，Debian/Ubuntu 下已得到最全面的测试验证。后续用到的编译测试工具大部分已经预编译打包，仅需一条 `apt-get` 命令就能快速安装。

编译器

目前仅支持 ARM CORTEX-M3/4 系列的单片机，用的是交叉编译器 `arm-none-eabi-gcc`，Ubuntu 的软件库已经集成，也可以自行编译。

编译链接还需要用到的 `Newlib C` 库，也已经集成在系统软件库中。

项目管理

与众多开源项目不同，MCUSH 的项目管理没有选用传统的 `make`，而是改用 `scons`，原因主要有：

- 1、作者熟悉 Python，使用 Python 语法的配置文件更得心应手。
- 2、需要搭配自行开发的 `scons` 扩展库，才能灵活配置 MCUSH 的定制选项。

扩展库下载地址：http://github.com/pengshulin/sites_scons.git

版本管理

首选 `git`，并且整个项目都托管在 `github.com`。

调试器

使用 GCC 的配套调试器 GDB，交叉编译版 arm-none-eabi-gdb。

其缺点是公认的交互界面不太友好，有众多的前端工具可以弥补这一点，比如作者正在使用的折中方案：cgdb，一个终端环境下用的 GDB 封装，支持语法着色。

```

949 lua_writestringerror("PANIC: unprotected error in call to Lua API (%s)\n"
950 lua_tostring(L, -1));
951 return 0; /* return to Lua to abort */
952 }
953
954
955 LUALIB_API lua_State *luaL_newstate (void) {
956 lua_State *L = lua_newstate(l_alloc, NULL);
957 → if (L) lua_atpanic(L, &panic);
958 return L;
959 }
960
961
962 LUALIB_API void luaL_checkversion_ (lua_State *L, lua_Number ver, size_t sz
963 const lua_Number *v = lua_version(L);
964 if (sz != LUAL_NUMSIZES) /* check numeric types */
/home/trees/src/github/mcush/liblua/lauxlib.c
#3 0x08012a2c in shell_run () at /home/trees/linkong/shell_lab/software/mcush/s
hell_core.c:869
#4 0x080157e8 in pxPortInitialiseStack (pxTopOfStack=0x8012a2d <shell_run+52>,
pxCode=0x20004e90, pvParameters=0x4) at /home/trees/linkong/shell_lab/software/l
ibFreeRTOS/portable/GCC/ARM_CM4F/port.c:214
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb)

```

GDB 远程调试需要代理服务，目前使用的是 OpenOCD，调试 ARM CORTEX-M 比较顺利。

集成开发环境

作者开发时避免使用大而全的 IDE（如 Eclipse），而是用众多小而精的工具组合。

如果你想使用自己熟悉的 IDE 开发，需要些额外工作，将上述工具链组合进去。

远程开发测试环境

由于上述开发环境都是基于命令行终端，很方便就能搭建一套远程开发测试环境：

- 1、在一台基于 LINUX 的开发机上安装全部的编译、调试、测试工具。
- 2、远程登录该机器，远程编辑、编译、调试和测试。

若局域网环境较好，首选这种方案，这样工程师电脑和开发测试电脑分离，相互不影响，如：

- 1、不会因为意外的断电重启影响当前的调试测试。
- 2、不会因为 USB 串口的插拔打乱分配到的串口编号（在 LINUX 下，USB 串口号是动态顺序分配的）。
- 3、不会因为要下班了而中断还在进行的测试环境。

如能实现外网远程登录（如搭建 VPN），开发测试人员的灵活性就更大了，甚至可以工程师在家里远程调试在公司的开发环境，或者出差在外远程检查一下测试机的设备日志。

远程登录容易遇到网络异常断开的情况，需要远端具有这种容错性，作者推荐使用 tmux，不仅解决上述问题，还支持终端分割、窗口管理，方便调试测试。

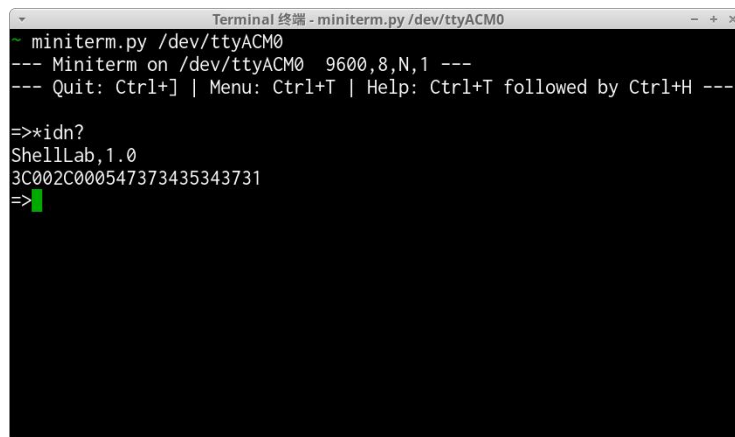
2、串口调试环境

Linux

终端环境下首推 python-pyserial 自带的调试器 miniterm.py，这是一个 pyserial 库的演示脚本，功能单一但已经够用了。

安装：sudo pip install pyserial，注意不要与 serial 库混淆。

安装完成后执行 miniterm.py <device_name>

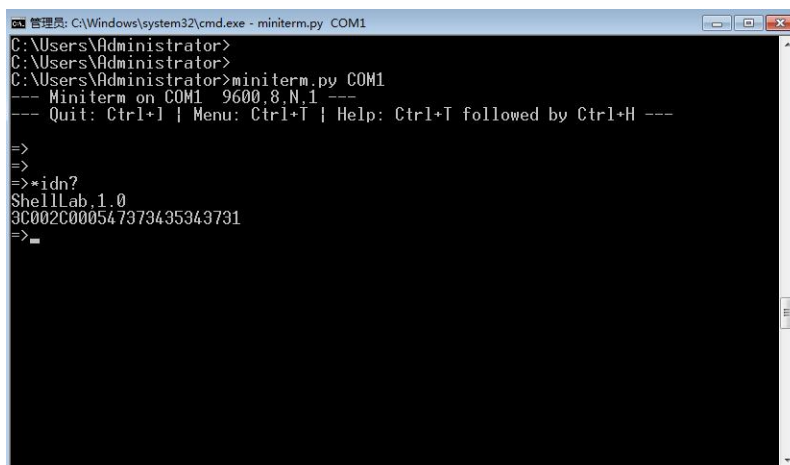


```
Terminal 终端 - miniterm.py /dev/ttyACM0
~ miniterm.py /dev/ttyACM0
--- Miniterm on /dev/ttyACM0 9600,8,N,1 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---

=>*idn?
ShellLab,1.0
3C002C000547373435343731
=>
```

Windows

上述的 miniterm.py 依然可用，但是受限于 Windows 自带的命令行终端，不太灵活。

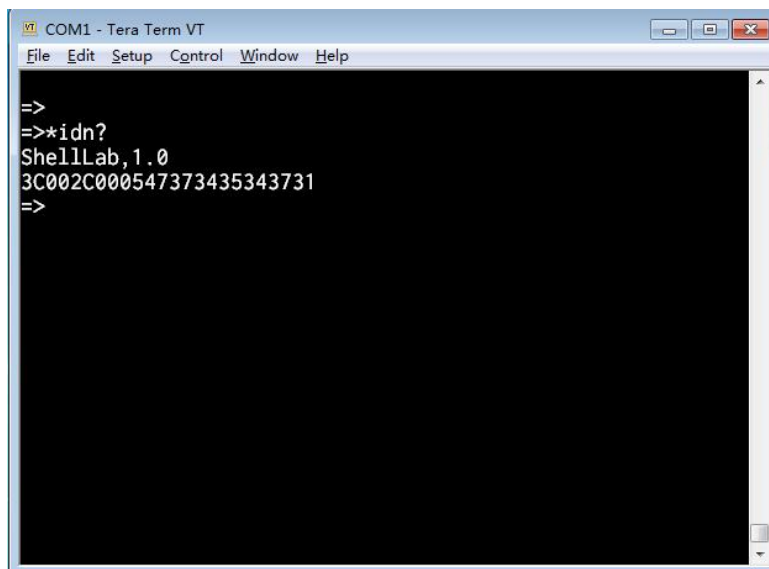


```
管理员: C:\Windows\system32\cmd.exe - miniterm.py COM1
C:\Users\Administrator>
C:\Users\Administrator>
C:\Users\Administrator>miniterm.py COM1
--- Miniterm on COM1 9600,8,N,1 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---

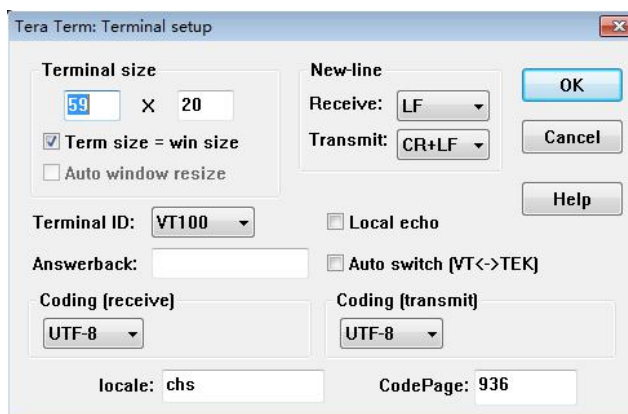
=>
=>
=>*idn?
ShellLab,1.0
3C002C000547373435343731
=>
```

首选的替代方案是 Tera Term，配置灵活、字体可调、支持行编辑的图形工具。

下载：<https://tssh2.osdn.jp/>



需要注意的是：设置新行结束符 LF（收）和 CR+LF（发），如下图。



3、自动测试环境

Python 库

MCUSH 提供封装好的模块库，通过调用串口，模拟手动调试指令，读取分析返回结果，并提供更多 API，帮助搭建测试脚本和工具。

安装方式：sudo pip install mcush

图形化工具

有时需要构建图形化应用，提供更方便直观的工具，降低使用门槛。

过程大致都是先用命令行测试脚本将需求基本验证完成后，再用图形库封装一下；如果需要部署到 Windows 机器上运行，又有代码保密性要求，就还需要发布成可执行的 EXE 和并打包成安装程序。

作者常用的图形封装库是 wxPython，绘制对话框的设计工具是 wxGlade。

