Environmental Sample Processor Software Environment Variables and File Directory Layout

1/4/10 Brent Roman   brent@mbari.org

Unix Command Shells

- All are interpreted, dynamic scripting languages
    - Optimized for starting and managing other processes
    - Which may in turn be other command shells
    - Input may come from user at a terminal or canned script text files
- From *“ash” to “zsh”* -- many, more or less compatible alternatives

- “sh” -- the original Bourne shell (by Steve Bourne while at Bell Labs)
- “csh” -- 'C'-like, improved on tcsh (by Bill Joy while at UC Berkeley)
- “ksh” -- Kron shell (by David Korn of Bell Labs)
- “bash” -- Bourne Again SHell: GNU's answer to “sh”
  - Big, Bloated and Slow with lots of cool, mind bending features
  - Default shell on most desktop Linux distros where RAM is plentiful
  - And, you OS/X mac heads know it as the “terminal window”
- “rush” -- the RUby SHell: a command shell written in Ruby
- “zsh” -- the Z SHell
  - Attempts a synthesis of those that came before
- “ash” -- the A SHell (by Kenneth Almquist)
  - Small and very compatible with bash
  - Used in many memory constrained or embedded Linux products
    - Wi-Fi routers
    - Set up boxes
    - And, our own ESP !!

Environment Variables

- Each program (or process) runs in an “environment” consisting of:
  - Command Line arguments passed explicitly after the command name
  - And, environment “variables” or keys associated with text values
    - It's easy to create a new one or alter an existing one's value
      FOO=BAR
    - Most shells use $ prefix to replace environment variable's name with its text value
      echo $FOO ==> writes “BAR”
    - Environment variables marked for export are inherited from parent process
      export FOO=bar; sh -c 'echo $FOO' ==> writes “bar”
    - Or, they may be passed into a single process much like a command argument
      FOO=bar sh -c 'echo $FOO' ==> also writes “bar”
      echo $FOO ==> writes “BAR”
  - The *env* command lists all environment variables

- Common environment variables
  - *HOME*=Current working directory
    - Changed with the *cd* shell built-in command
  - *PATH*=Colon separated list of directories to search for executable files with no leading /
  - *USER*=User's login name
  - *DISPLAY*=machine:screen# (where X-windows sends graphics)
    typically *localhost:0* # the first local screen

Propagating Environment Variables

- Processes can change their own environments
  - But, ***they cannot change the environments of others***
  - Shell scripts that change environment variables don't have any lasting effect

    sh -c "FOO=notBAR"; echo $FOO ==> writes "BAR"

    sh -c "cd /"; pwd => writes "/home/brent" not "/"

    - **UNLESS** they run in the same shell process
      - *FOO=notBAR; echo $FOO* ==> writes "notBAR"
  - Shell built-in commands run without creating a new process
    - As opposed to external commands
    - The cd command MUST always be implemented as a shell built-in
      - Because it changes *$HOME*
    - The source or '.' commands run a file of commands through the current shell process
      - No subshell is created
      - So scripts can affect the current shell's environment when desired
    - Some other commands are built-ins for speed given their frequency of use
      - Creating Unix processes is relatively slow and memory intensive
    - User written programs are always external commands
      - But user written scripts may be sourced without creating a new shell

ESP Environment Variables

- *ESPhome* is top level (root) directory of ESP source code tree
  - Default *ESPhome=$HOME/esp2*

- *ESPname* is the name of the ESP machine
    - Determines command prompt and which configure.rb to read
    - Change to masquerade as another ESP machine or for desktop simulations
  - Default *ESPname=`hostname`* with any "ESP" prefix removed

- *ESPmode* is the operating mode in which to run the ESP software
  - Default *ESPmode=real*
    - "real" means real-time with real hardware
    - "simfast" means fast as possible with simulated hardware
    - "simreal" means real-time with simulated hardware
    - "quick" is like simfast, but with minimal console log messages
  - These and more are defined in directory *$ESPhome/mode* as short ruby (.rb) script files
  - To run esp once in "quick" (simulation verification) mode:
    - *ESPmode=quick esp aMissionScriptName*

- *ESPlog* is the root directory under which all data files are written
  - Default *ESPlog=/var/log/$USER*
  - Esp software normally does not write into the *$ESPhome* source code tree
  - For simulation on desktop, one must grant *$USER* access to */var/log/$USER* directory
    - Or set ESPlog to something under user's home. e.g.   *$HOME/espLog*

ESP Environment Variables and Configuration Files

- *ESPpath* is a list of directories to search for mission scripts
  - Default *ESPpath=.:$ESPhome/mission:$ESPhome/protocol*

- *ESPconfigPath* is a list of directories to search for configuration files
  - *ESPconfigPath=$ESPhome/espType/$ESPname:$ESPhome/espType:$ESPhome/admin*
  - EspType is either *shallow*, *mfb*, *1km*, or *4km*
    - All espTypes are configuration subdirectories under *$ESPhome* containing:
      - *initialize.rb* to configure serial communication ports
        - Baud rates, stop bits, Unix port names (e.g. /dev/I2Cgate)
      - *netconfig.rb* to map dwarf objects to their real I2C addresses and log monikers
        - Also configures I2C gateways (retries, type of CRC protocol, etc.)
      - *preconfig.rb* defines objects that should be machine independent
        - e.g. Rotary Valve layouts, solenoids, basic camera config
      - *$ESPname/configure.rb* defines objects whose details are always machine specific
        - Changes can affect only machine $ESPname
      - *postconfig.rb* defines objects that may be machine specific
        - If they are missing on configure.rb, they get a default definition in postconfig
          - e.g. Valve plumbing, tweaks for puck handling
      - Be very careful when modifying shared configuration files
        - It's easy to make your machine work while breaking another!

Ruby Environment Variable

- *RUBYLIB* is a list of directories to search for “required” Ruby libraries and scripts
  - Typically *RUBYLIB=$ESPhome/lib:$ESPhome/utils:$ESPhome/protocol*
- Only *require* “file” uses *$RUBYLIB*
  - *require* is a core Ruby method
- *define* or *execute* “file” use $ESPpath
  - Because *define* and *execute* are ESP specific additions to Ruby

ESP Source Code Tree Executables

- All directories live under *$ESPhome* (usually */home/$USER/esp2*)
  - *.../bin* contains executable scripts that may be invoked from the Unix shell
    - Some are implemented as shell scripts, others are Ruby scripts used as commands
      - *esp*, *espclient*, *showlog*, etc.
    - The *ESPenv* script automatically assigns ESP environment variables
      - Recall *ESPenv* must be “sourced” into the current shell with '.' or 'source' built-ins
      - Usually sourced (read and executed) automatically in the shell's *.profile* script
        - .profile is automatically sourced by bash and ash when they are started
        - File names beginning with dot are hidden
          - View them with ls -a #list all files
      - All arguments to *ESPenv* are optional
      - 1st argument is the value for ESPname
        - Defaults to *hostname*
      - 2nd argument is the type of esp deployment (i.e. the espType)
        - *mfb*, *shallow*, *1km*, or *4km*
      - To simulate a mission on ESPgordon attached to the 4km DWSM:
        - . *ESPenv 4km gordon* #don't forget the leading dot

ESP Source Code Tree
Core Ruby Libraries

- *$ESPhome/lib* contains core Ruby libraries
  - *.../i2c* contains low-level Ruby scripts to handle I2C bus messaging
  - *.../dwsm* contains primitives for handle the DWSM dpress board and sample bags
  - *.../elmo* contains primitives for driving Elmo motor controllers via RS-232 cmds
    - Used only in the (now obsolete) 1km DWSM
  - *.../gauge* contains primitives to drive simple sensors via RS-232
    - For now, just the 4km DWSM's Stellar digital pressure gauges
    - These are core, not contextual, sensors
  - *.../instrument* contains contextual sensor drivers (and PCR ?)
    - CTD & ISUS
    - PCR is here too, but that's mainly because Bob Herlien wrote it.
  - *.../posix* Generic drivers for “normal” serial ports
    - As opposed those accesses via Dwarves, which are found in *.../i2c*

ESP Source Code Tree
Core Ruby Libraries and Utilities

- *$ESPhome/lib* contains core Ruby libraries and hardware drivers
  - Scheduler, Delay, Threads, Log, Slide, Shaft, Solenoid, Thermal, Clamp, Camera...
  - *.../i2c* contains low-level Ruby scripts to handle I2C bus messaging
  - *.../dwsm* contains primitives for handle the DWSM dpress board and sample bags
  - *.../elmo* contains primitives for driving Elmo motor controllers via RS-232 cmds
    - Used only in the (now obsolete) 1km DWSM
  - *.../gauge* contains primitives to drive simple sensors via RS-232
    - For now, just the 4km DWSM's Stellar digital pressure gauges
    - These are core, not contextual, sensors
  - *.../instrument* contains contextual sensor drivers (and PCR ?)
    - CTD & ISUS
    - PCR is here too, but that's mainly because Bob Herlien wrote it.
  - *.../posix* Generic drivers for “normal” serial ports
    - As opposed those accesses via Dwarves, which are found in *.../i2c*

- *$ESPhome/utils* contains common utilities (directly above core libraries)
  - romanFlush, calarm, calcar, puckmoves, (shallow) sampler, shuffle, etc.
  - *.../dwsm* contains utilities for 1km DWSM
    - *.../4km* contains utilities specifically for 4km DWSM

ESP Source Code Tree
Science Protocols

- *$ESPhome/protocol* contains Ruby code implementing science assays
  - These scripts are intended to be modified by investigators
  - *BAC*, *HAB*, *LARV*, *wcr*, etc.
  - *sh2* common to all sandwich hybridization assays
  - *sh1* common to most assays that collect samples or make lysate
  - Utilities common between most assays
  - *pcrslug*, *spe*, *shortmfb* for PCR
  - *DA* and *DAprocess, PRVprocess* for Demoic Acid detection

ESP Source Code Tree
Mission Scripts

- *$ESPhome/mission* contains top-level scripts that control mission behavior
  - YyMonthDDname missions
    - These scripts are written the day before deployment :-)
  - *skeleton* mission primitives
    - Defines the general behavior of all missions
    - Also implements simulation behaviors for protocols
      - Until simulation is (properly) pushed completely into core libraries
  - *dwsm4km* mission primitives for 4km DWSM
    - Augments *skeleton* for DWSM
  - *phasecfg* configures mission parameters
    - Where to send email messages
    - How to configure contexual sensors
    - Default sample volumes and camera parameters for each assay type

ESP Log File and FTP site Directory Layout

- /var/log == ftp://espName == top directory of the FTP site
  - Don't use the Mac “finder” to browse an ESP's FTP site
    - Too much traffic generated for file previews
    - Use Firefox or Cyberduck instead.
  - Windows MS Explorer is also fine

  - */var/log/messages* contains Linux kernel messages
  - */var/log/vsftpd.log* records all FTP site traffic
  -
  - Kernel log files grow continuously
    - Empty (or truncate) them as user root with
      - # > /var/log/messages
      - # > /var/log/vsftpd.log
    - Do not remove them with the *rm* command

  - */var/log/$USER* is top level output directory for each user's ESP data
    - May be overridden with *$ESPlog* environment variable
    - But moving it may remove it from the FTP site hierarchy
    - These files are owned by *$USER*, not user root

ESP Output Directories

- Actual names and contents determined by settings in configuration files
- By default:
  - *.../hires*
    - High resolution camera images (each approx. 3.5 Mbytes)
  - .../lores
    - Low resolution camera images
  - .../midres
    - Medium resolution camera images
    - Typically auto-exposures
  - Top directory contains “default” resolution camera images
    - Typically fixed exposures

- Only files in the top directory are automatically uploaded to shore servers
    - To conserve radio link bandwidth
    - You may upload selected files is hires or other subdirs manually via scp
      - If the radio link is of good quality and will not be busy for a while

MBARI

ESP Output File Types

- **.tif*   =   TIFF camera images
  - Tagged Image File Format
  - Examine with ImageJ from http://rsbweb.nih.gov/ij/
  - It's a nice idea to install imageJ as a “helper” app for TIFFs in your web browser
- **.pcr* = Comma Separated Value PCR data
  - Formated for direct input into Excel or similar spreadsheet
  - May also be viewed in a text editor
- **.out* = console output capture
  - Text normally output to the esp interactive console
  - Redirected here when running non-interactive mission script
    - i.e. When esp is run with:
      - *start esp* YymonthDayScript
    - May be monitored with the *showlog* command
      - Or viewed in any text editor
    - File name is the operating mode as in **.log* below
- **.log* = detailed, binary esp engineering logs
  - File name is the operating mode
      - real.log is the one with real hardware
      - quick.log is from quick simulation runs
      - May not be viewed on a text editor
      - View with the Ruby *dumplog* command