```
#!/bin/sh
####!/bin/bash   ### switched so it's also valid in busybox (ash) shell
#   script header with basic functions
# Author: joe brendler 19 Jul 2014
#         24 June 2025 - v1.0.0 moved to /usr/sbin/ and consolidated as script_header_joe
too
#
toc() # table of contents (this fn just calls summarize_header
{ # call summarize_header
summarize_header; printf "\n"
message "That's all I have to say about that"
}
# @usage source /usr/sbin/script_header_joetoo
# @deps package dependencies are managed by the dev-util/script_header_joetoo ebuild
# @req linux shell
# @note all functions in script_header_joetoo are designed to work in any POSIX shell
# @note other script header modules are available as USE flag selections in the dev-util/
script_header_joetoo package
#-----[ metadata tagging key ]-----------------------------------------------
# @note
# @note lines below starting '# @xxx' are interpreted by summarize_header as metadata
# @usage @usage describes the usage syntax, options, and arguments for a function
# @args @args defines positional parameters ($1, $2, etc) and their roles
# @vars @vars identifies global variables required or modified by the function
# @ret @ret identifies value(s) returned and/or emitted
# @deps @deps lists function, script, or package dependencies
# @req @req specifies prerequisites or state requirements (e.g., must be root)
# @rule @rule outlines logic constraints or mandatory behaviors
# @note @note provides administrative or general notes for the user or maintainer
# @warn @warn provides administrative warning for user or maintainer
# @ex @ex provides a usage example (standardized for single-line compatibility)
# @cont @cont identifies a continuation of the previous metadata tag


#=====[ Section 1: Global Variables  ]============================================
# @vars SCRIPT_ENV: script paths and environment state (logFile, VERBOSE, TRUE/FALSE)
# @vars MSG_STRINGS: standardized error messages (E_NOTROOT, E_BAD_ARGS) and UI tokens (n
o_msg)
# @vars REGEX_POSIX: portable BRE patterns (W0/W1 whitespace, P0/P1 printable, C0/C1 cont
rol)
# @vars ANSI_SGR: "Pre-cooked" SGR styling (RED, GRN, BOLD) and CSI control initiator
# @vars DEC_GRAPHICS: DEC Special Graphics codes (H_wall, V_wall) and set toggles (SO/SI)
# @note Variables use CAPITALIZED names; ANSI sequences are binary "pre-cooked" for print
f '%s' compatibility

BUILD="0.0.0 (19620207)"  #redefine in base script
ROOT_UID=0       # Only users with $UID 0 have root privileges
#script_header_installed_dir="/usr/sbin"
script_header_installed_dir=/home/joe/myUtilities/dev-util/script_header_joetoo
#script_header_installed_dir=/home/joe/myUtilities/dev-util/script_header_joetoo/testing

non_interactive_header="${script_header_installed_dir%/}/script_header_joetoo_noninteract
ive"

# define default logFile (used by all logging functions)
#   this is a global variable calling scripts SHOULD define themselves,
#   so this is really just a safeguard - not a mandate on where to log
# note: using id -u here to test if the user is root, b/c this is the top
#   of the header where no functions are defined yet, but once defined,
#   isroot() can do the same sort of thing
if [ "$(id -u)" -eq 0 ]; then
  logFile=/var/log/script_header_joetoo.log
else
  # Fallback to home dir or /tmp if user sourcing header is not root
  logFile="${HOME:-/tmp}/script_header_joetoo.log"
fi

# use id -un to assign a default value for the global $user variable
#   this is a global variable calling scripts SHOULD define themselves,
#   so this is really just a safeguard - not a mandate on where to log
```

```
user=$(id -un)

#  Error message used by various scripts
E_NOTROOT="Must be root to run this script"
E_ROOT="Please run this script as user, not root"
E_BAD_ARGS="Improperly formatted command line argument"
E_BAD_OPT="Invalid option"
E_BAD_COLOR="Invalid color argument"

# pseudo-boolean values for most shells which have none
TRUE=0    # will evaluate to be logically true in a boolean operation
FALSE=""  # will evaluate to be logically false in a boolean operation
  # play with test function:
  # test() { [ $1 ] && echo "$1 is true" || echo "$1 is false"; }
# in comparison, exit status code must be tested with math, not logic
  # play with test function:
  # result=1; [ "$result" ] && echo true || echo false  {status 1 == failure; but this wi
ll return "true"}
  # to test an exit status, use [ "$result" -eq 0 ]

# no_msg is a space overwritten by a backspace, used log_handle_result() to send null mes
sage with a non-null string
no_msg=" \b"

# check set null vs unset and assign default VERBOSE and verbosity,
#  but only if not yet assigned by calling program
VERBOSE="${VERBOSE:-$FALSE}" # Set default ($FALSE) if VERBOSE is UNSET or NULL (already
"$FALSE")
verbosity="${verbosity:-2}"  # Set default (2) if verbosity is UNSET or NULL

#-----[ regex character class patterns ]------------------------------
W0="[[:space:]]*"     # POSIX regex match for zero+ whitespaces
W1="[[:space:]]${W0}" # POSIX regex match for one+ whitespaces ([[:space:]]+ is not porta
ble)
P0="[[:print:]]*"     # POSIX regex match for zero+ printable characters (incl. whitespac
e)
P1="[[:print:]]${P0}" # POSIX regex match for one+ printable characters (incl. whitespace
)(+ not portable)
G0="[[:graph:]]*"     # POSIX regex match for zero+ printable characters (excl. whitespac
e)
G1="[[:graph:]]${G0}" # POSIX regex match for one+ printable characters (excl. whitespace
)(+ not portable)
A0="[[:alnum:]]*"     # POSIX regex match for one+ alphanumeric characters
A1="[[:alnum:]]${A0}" # POSIX regex match for one+ alphanumeric characters(+ not portable
)
C0="[[:cntrl:]]*"     # POSIX regex match for zero+ control characters
C1="[[:cntrl:]]${C0}" # POSIX regex match for one+ control characters
F1='^[_[:alpha:]][_[:alnum:]]*'   # regex match for one+ characters allowed in POSIX func
tion name
BF1='^[_[:alpha:]][[:alnum:]_-]*' # regex match for one+ characters allowed in Bash funct
ion name
# The definitions above conform to the "lowest common denominator"
# Basic Regular Expression (BRE) standards rather than using
# Extended Regular Expressions(ERE), to make them more portable
# (e.g. will work with any sed, because default is BRE and would
#  require use of an extra -E flag to work if we defined to a ERE standard)

#-----[ globals supporting terminal io ]------------------------------------------
# Easy ANSI Escape sequences to put color in my scripts ]---
#   see http://en.wikipedia.org/wiki/ANSI_escape_code
#   see also http://ascii-table.com/ansi-escape-sequences.php

#-----[ ASCII Control Characters ( [:control:] )]---------------------------
#Octal  Hex     Abbr    Description
#000    00      NUL     Null
#001 - 006      01-06   SOH-ACK Start of Heading to Acknowledge
#007    07      BEL     Bell (audible alert)
#010    08      BS      Backspace
#011    09      HT      Horizontal Tab (\t)
#012    0A      LF      Line Feed / Newline (\n)
```

```
#013    0B      VT      Vertical Tab
#014    0C      FF      Form Feed (\f)
#015    0D      CR      Carriage Return (\r)
#016 - 032      0E-1A   SO-SUB  Shift Out to Substitute; note S0 & S1 ar "pre-cooked" bel
ow
#033    1B      ESC     Escape (Start of your ${CSI})
#034 - 037      1C-1F   FS-US   Separators
#177    7F      DEL     Delete
# potentially useful in scripts --
NUL='\000'  # only for matching  (do not try to print)
BEL='\007'  # only when you literally need bells and whistles
BS='\010'   # note: this is non-destructive (not a delete)
HT='\011'   # equiv \t
LF='\012'   # equiv \l
VT='\013'   #
FF='\014'   # equiv \f
CR='\015'   # equiv \r
ESC='\033'  # see CSI, ESCon, ESG0
DEL='\177'  # delete
EOF='\032'  # End of File (aka SUB)

# (POSIX) CSI - define the control sequence initiator: ascii "\e[", octal "\033[", or hex
 "\x1b["
# which is used for Parameterized Control (e.g., cursor positioning, setting colors, clea
ring screen)
# The sequence is always followed by numbers and a final letter (e.g., \e[31m for red tex
t)
# LEGACY: assigning CSI='\033[' stores the characters of the "Literal Escape Sequence" in
 a string
#   requires an interpreter (echo -e or printf %b) to convert the '\' '0' '3' '3'
#   strings into a real control character
# UPGRADED: assigning CSI=$(printf '\033[') stores the "Raw ANSI Control Sequence"
# The subshell executes and emits the actual non-printing Escape Byte (Hex 1B)
# This is "Pre-Cooked" -- it works in printf '%s' because the byte is already binary
# Note: using octal \033 rather than hex \x1b because octal remains the gold standard
#   for portable shell scripts... specifically, POSIX explicitly requires support for
#   octal escapes (\ddd). It does not require support for hexadecimal escapes (\x),
#   Many strictly POSIX shells (like dash or certain minimalist versions of ash) will
#   treat \x1b as literal text, and while some specific BusyBox or Alpine Linux versions
#   of ash might have expanded printf to include \x, this is a non-standard extension;
#   relying on it (\x1b) makes the script less portable
CSI=$(printf "${ESC}[")

#-----[ Select Graphics Rendition on/off ]----------------------------------------
BOLD="1"          # bold on
UL="4"            # underline on
BLINK="5"         # slow blink on
BLINKFAST="6"     # fast blink on
REVERSE="7"       # image: inverse (reverse video)
ULoff="24"        # underline off
BLINKoff="25"     # blink off
SGRoff="0"        # Bold off (reset all SGR (e.g. blink, underline)
#-----[ Set Text Color, Foreground ]----------------------------------------------
BLACK="30"        # foreground black
RED="31"          # foreground red
GREEN="32"        # foreground green
YELLOW="33"       # foreground yellow
BLUE="34"         # foreground blue
MAG="35"          # foreground magenta (it's like fucia)
CYAN="36"         # foreground cyan
LBLUE="36"        # foreground light blue (cyan)(deprecated - keep here for compatibility)
WHITE="37"        # foreground white
FG_DEFAULT="39"   # foreground to terminal default
#FG_SPEC="38"     # foreground to specified (e.g., 2 for RGB or 5 for a 256-color palette)
# For example, ${CSI}38;2;255;165;${SGRoff}m sets the foreground to orange
# NOT USED HERE. used in RGB_FG() in script_header_joetoo_extended
#-----[ Set Background Color ]----------------------------------------------------
BACKoff="40"      # background black
RBACK="41"        # background red
GBACK="42"        # background green
```

```
YBACK="43"       # background yellow
BBACK="44"       # background blue
MBACK="45"       # background magenta
CBACK="46"       # background cyan
LBACK="46"       # background light blue (cyan)(deprecated - keep here for compatibility)
WBACK="47"       # background white
BG_DEFAULT="49"  # foreground to terminal default
#BG_SPECB="38"   # background to specified (e.g., 2 for RGB or 5 for a 256-color palette)
# For example, ${CSI}48;2;0;0;${SGRoff}m sets the background to black
# NOT USED HERE. used in RGB_BG() in script_header_joetoo_extended

#-----[ My Favorite Colors (terminate with ${Boff} ]---------------------------------
Ron="${CSI}${RED}m"
Gon="${CSI}${GREEN}m"
Yon="${CSI}${YELLOW}m"
Bon="${CSI}${BLUE}m"
Mon="${CSI}${MAG}m"
Con="${CSI}${CYAN}m"
Lon="${CSI}${LBLUE}m"   # (deprecated - keep here for compatibility)
Won="${CSI}${WHITE}m"
BRon="${CSI}${RED};${BOLD}m"
BGon="${CSI}${GREEN};${BOLD}m"
BYon="${CSI}${YELLOW};${BOLD}m"
BBon="${CSI}${BLUE};${BOLD}m"
BMon="${CSI}${MAG};${BOLD}m"
BCon="${CSI}${CYAN};${BOLD}m"
LBon="${CSI}${LBLUE};${BOLD}m"   # (deprecated - keep here for compatibility)
BWon="${CSI}${WHITE};${BOLD}m"
RVon="${CSI}${REVERSE}m"
Boff="${CSI}${SGRoff}m"          # Bold off (reset all SGR (e.g. blink, underline)
#-----[ Here setting background as well ]-----------------------------------------
BBonY="${CSI}${BLUE};${YBACK};${BOLD}m"

#-----[ for "localized" use ]-----------------------------------------------------
# when differentiation from above is needed, such as the "command preview"
# technique used in demonstrate_header
_BRon="${BRon}" ; _BGon="${BGon}" ; _BYon="${BYon}" ; _BBon="${BBon}"
_BMon="${BMon}" ; _BCon="${BCon}" ; _BWon="${BWon}" ; _Boff="${Boff}"
_Ron="${Ron}" ; _Gon="${Gon}" ; _Yon="${Yon}" ; _Bon="${Bon}"
_Mon="${Mon}" ; _Con="${Con}" ; _Won="${Won}"
#-----[ for "theme" engines ]-----------------------------------------------------
# primary use: construct "theme engine" for the demonstration suite
# matching common regex-based syntax highlighting rules
_cmd_color="${BBon}"   # command
_var_color="${BRon}"   # variable (e.g. $var)
_quote_color="${BYon}" # quotation (e.g. "..." or '...'
_opt_color="${BMon}"   # option (e.g. -v)
_cmt_color="${Con}"    # comment
_data_color="${Mon}"   # data (e.g. 0.5)
_ctl_color="${Gon}"    # control keyword (e.g. if, while, do, case ...)
_func_color="${BGon}"  # functions (e.g. myFunction()
_op_color="${Gon}"     # operators (e.g. &&, ||, |, >)

#-----[ globals supporting Box Drawing methods ]--------------------------
# See: https://en.wikipedia.org/wiki/Box-drawing_character
# Note: this is from Digital Equipment Corporation (DEC) command language (DCL) experianc
e
# I had at Nordson Corporation (Amherst, OH) Robotics EE Research department in 1982
# (though that was actual VT102 terminals connected by serial lines to a Vax-11/70 mainfr
ame)
# DEC was the pioneer that developed the VT100 terminal in the late 1970s. DCL programs f
or the VT100
# used of escape sequences (\e or ESC) to control cursor movement, colors, and special ch
aracter sets,
# and this became the industry standard. Most modern terminal emulators (like xterm, Kons
ole, or the
# console in BusyBox) still implement compatibility with these original VT series termina
ls
#
# Define the escape sequence initiator (ESCon) not to be confused with the CSI defined ab
```

```
ove
# with a Single Shift Unlock (SSU) followed by a character set selection.
# The SSU is: ascii "\e(", octal "\033(", or hex "\x1b(" ,  and appending character "0" m
akes ESCon
# a Character Set Selection that tells the terminal to switch the current font mapping
# (the G0 character set) from the standard ASCII font to the DEC Special Graphics font.
# (VT100-compatible terminals maintain four character sets, known as G0, G1, G2, and G3 -
#  G0 is the primary character set. It's what the terminal uses by default for all incomi
ng
#  characters; G1-G3 are secondary sets)
# ESCon and ESCoff are thus SSU sequences that permanently map G0 to another G-set until
# another sequence shifts it again

# ESC(C Sequences (Explicit G0 Selection)
ESCon=$(printf "${ESC}(0")   # Maps G0 to DEC Special Graphics
ESCoff=$(printf "${ESC}(B")  # Maps G0 to Standard ASCII
# ESC)C Sequence (Explicit G1 Selection)
ESG1=$(printf "${ESC})0")    # Maps G1 to DEC Special Graphics
# Shift In/Out Control Characters (Fast G0/G1 Toggling)
SO=$(printf '\016')        # Shift Out (Activate G1, Graphics ON - ASCII 14, hex \x0E)
SI=$(printf '\017')        # Shift In (Activate G0, Standard ASCII ON - ASCII 15, hex \x0
F)
# note: migrated all of these from legacy string assignments (e.g. ESCon="${ESC}(0")
#   to subshell printf generation of the actual binary escape code bytes
#   (i.e. pre-cooked and don't need the -e for echo or the %b vs %s for printf)

# Graphics Character Codes (Using portable octal escapes)
H_wall='\161'      # Horizontal line (0x71)
V_wall='\170'      # Vertical line (0x78)
UL_wall='\154'     # Upper-Left corner (0x6C)
UR_wall='\153'     # Upper-Right corner (0x6B)
LL_wall='\155'     # Lower-Left corner (0x6D)
LR_wall='\152'     # Lower-Right corner (0x6A)
# note: these should NOT be pre-cooked like the control sequences above
#   b/c that would convert them to binary representation of letters (e.g. \161 => q)
#   and we need to presesrve the octal code so it cab be interpreted as graphic
#   after the interpreter (e.g. echo -e or printf '%b') has received the
#   control sequence telling it to do so

# unicode line/box characters also exist, but have been moved to
#   script_header_joetoo_unicode (do not require the esc seq intro)

#=====[ Section 2: Terminal & Cursor Control Functions ]=============================
====================
color() # (POSIX) set FG $1, optional [bold $2], optional [BG $3]
{ # @usage color FG [BOLD] [BG]
  # @args $1: FG code (30-37, 39); $2: Bold (0/1); $3: BG code (40-47, 49)
  # @ret Raw ANSI escape sequence (pre-cooked CSI...m)
  # @rule Validates inputs using case; returns E_BAD_COLOR and exit 1 on invalid codes
  # @ex color 31 1 44  # returns Red on Blue, Bold
  # @note Use $(color) with no args to reset to terminal default (equiv to Boff)
    _FG=""    # Foreground (3x)
    _BG=""    # Background (4x)
    _B=""     # Bold (0/1)
    _V_FG="${1:-39}"  # Default _FG is 39 (terminal default)
    _V_B="${2:-0}"    # Default Bold is 0 (Normal)
    _V_BG="${3:-49}"  # Default _BG is 49 (terminal default)
    # validate foreground (3x) or set default 39; set bold appropriately
    case "$_V_FG" in
        3[0-7])  # foreground is set
            _FG="$_V_FG" ;
            # validate bold (0/1)
            case "$_V_B" in
                1) _B=";1" ;;
                0) _B="" ;;
                *) echo "$E_BAD_COLOR: Bold attr ($_V_B)(0/1)"; return 1 ;;
            esac  # _B
            ;;
        39)     # unset foreground for default
            _FG="" ;
```

```
            # validate bold (0/1) - omit leading ; since no $1
            case "$_V_B" in
                0|1) _B="$_V_B" ;;
                *) echo "$E_BAD_COLOR: Bold attr ($_V_B)(0/1)"; return 1 ;;
            esac  # _B
            ;;  # allow unset args $(color) to equate to ${Boff}
        *) echo "$E_BAD_COLOR: Foreground ($_V_FG)(3x)"; return 1 ;;
    esac # _FG

    # validate background (4x) or set default (unset)
    case "$_V_BG" in
        4[0-7]) _BG=";$_V_BG" ;;
        49) _BG="" ;;  # allow unset _BG to imply terminal default
        *) echo "$E_BAD_COLOR: Background ($_V_BG)(4x)"; return 1 ;;
    esac
    # output the raw ANSI escape sequence
    printf "%s%s%s%s%s" "${CSI}" "${_FG}" "${_B}" "${_BG}" "m"
    # Clean up local variables to maintain POSIX hygiene
    unset -v _FG _BG _B _V_FG _V_B _V_BG
}

termwidth() # (POSIX plus) calculate and output the width of the terminal
{ # @usage termwidth
  # @ret Integer representing number of terminal columns
  # @rule Priority: 1. $COLUMNS env; 2. tput cols; 3. stty size; 4. Default (80)
  _cols="$COLUMNS"   # prefer environment variable; if null try tput; or fall back to stt
y
  [ -z "$_cols" ] && _cols=$(tput cols 2>/dev/null)
  [ -z "$_cols" ] && _cols=$(stty size < /dev/tty 2>/dev/null | cut -d' ' -f2)
  printf '%s' "${_cols:-80}"   # fall back to default (standard) 80 cols if none of above
 worked
  unset -v _cols ; }

termheight() # (POSIX plus) calculate and output the height of the terminal
{ # @usage termheight
  # @ret Integer representing number of terminal lines
  # @rule Priority: 1. $COLUMNS env; 2. tput cols; 3. stty size; 4. Default (80)
_rows="$LINES"   # prefer environment variable; if null try tput; or fall back to stty
  [ -z "$_rows" ] && _rows=$(tput lines 2>/dev/null)
  [ -z "$_rows" ] && _rows=$(stty size < /dev/tty 2>/dev/null | cut -d' ' -f1)
  printf '%s' "${_rows:-25}"   # fall back to default (standard) 25 lines if none of abov
e worked
  unset -v _rows ; }

vt_init() # (POSIX) Initialize G0/G1 character sets for DEC SI/SO graphics
{ # @usage vt_init
  # @note maps G0 to ASCII and G1 to DEC Special Graphics
  # @cont this enables the use of SO (\016) and SI (\017) for fast toggling
  # @cont between modes, which is faster than ESCon/off
  # ( ensures standard ASCII ('B') is mapped to G0 (primary display set)
  #   maps DEC Special Graphics ('0') to G1 (secondary set)
  #   this is what prepares the terminal for the fast SI/SO switching )
  printf "%b%b" "$ESCoff" "$ESG1"
}

#---[ Simple Cursor State Control ]-----------------------------------
SCP() # (ANSI) save the current cursor position
{ printf '%b' "${CSI}s" ; return $?; }
# @usage SCP
# @ret 0 on success (sequence emitted)
# @note Positions are terminal-internal; not stored in shell variables

RCP() # (ANSI) restore the cursor to the saved position
{ printf '%b' "${CSI}u" ; return $?; }
# @usage RCP
# @ret 0 on success

HCU() # (ANSI) Hide the cursor (Note: the trailing character is lowercase L)
{ printf '%b' "${CSI}?25l" ; return $?; }
# @usage HCU
```

```
# @ret 0 on success

SCU() # (ANSI) Show the cursor
{ printf '%b' "${CSI}?25h" ; return $?; }
# @usage SCU
# @ret 0 on success

CLR() # (ANSI) Clear stdout
{ printf '%b' "${CSI}2J" ; return $?; }
# @usage CLR
# @ret 0 on success

EL() # (ANSI) Erase line
{ printf '%b' "${CSI}K" ; return $?; }
# @usage EL
# @ret 0 on success

#---[ Cursor Absolute Positioning ]-----------------------------------
HVP() # (ANSI) move cursor to position row=$1, col=$2 (both default to 1 if omitted)
{ _row="${1:-1}"; _col="${2:-1}"; if isint "$_row" && isint "$_col"; then
    printf '%b' "${CSI}${_row};${_col}f" ; unset -v _row _col; return 0;
  else unset -v _row _col; return 1; fi }
# @usage HVP [row] [col]
# @args $1: Row (default 1); $2: Column (default 1)
# @ret 0 if moved; 1 if args are not integers
# @rule Validates input via isint()

CUP() # (ANSI) move cursor to position row=$1, col=$2 (both default to 1 if omitted)
{ _row="${1:-1}"; _col="${2:-1}"; if isint "$_row" && isint "$_col"; then
    printf '%b' "${CSI}${_row};${_col}H" ; unset -v _row _col; return 0;
  else unset -v _row _col; return 1; fi }
# @usage CUP [row] [col]
# @args $1: Row (default 1); $2: Column (default 1)
# @ret 0 if moved; 1 if args are not integers
# @rule Validates input via isint()

#---[ Cursor Relative Movement ]--------------------------------------
CUU() # (ANSI) Move the cursor up ($1 cells)
{ _reps="${1:-1}"; if isint "$_reps"; then
    printf '%b' "${CSI}${_reps}A" ; unset -v _reps; return 0;
  else unset -v _reps; return 1; fi; }
# @usage CUU [count]
# @args $1: Number of cells to move (default 1)
# @ret 0 if moved; 1 if arg is not an integer

CUD() # (ANSI) Move the cursor down ($1 cells)
{ _reps="${1:-1}"; if isint "$_reps"; then
    printf '%b' "${CSI}${_reps}B" ; unset -v _reps; return 0;
  else unset -v _reps; return 1; fi; }
# @usage CUD [count]
# @args $1: Number of cells to move (default 1)
# @ret 0 if moved; 1 if arg is not an integer

CUF() # (ANSI) Move the cursor fwd/right ($1 cells)
{ _reps="${1:-1}"; if isint "$_reps"; then
    printf '%b' "${CSI}${_reps}C" ; unset -v _reps; return 0;
  else unset -v _reps; return 1; fi; }
# @usage CUF [count]
# @args $1: Number of cells to move (default 1)
# @ret 0 if moved; 1 if arg is not an integer

CUB() # (ANSI) Move the cursor back/left ($1 cells)
{ _reps="${1:-1}"; if isint "$_reps"; then
    printf '%b' "${CSI}${_reps}D" ; unset -v _reps; return 0;
  else unset -v _reps; return 1; fi; }
# @usage CUB [count]
# @args $1: Number of cells to move (default 1)
# @ret 0 if moved; 1 if arg is not an integer

box_esca() # (POSIX) DEC box drawing usingw ESCon/off (Explicit G0 mapping)
```

```
{ # @usage box_esca "text"
  # @args $1: String to be boxed
  # @note Uses Explicit G0 mapping (slower but state-safe)
  # @note text is displayed inside a single-wall box
  _be_text="$1"
  _be_len=${#_be_text}   # length of text determines width of box
  # use fixed width printf to create proper length string of spaces | tr into wall segmen
t
  _be_h_line=$(printf "%*s" "$((_be_len + 2))" "" | tr ' ' "${H_wall}")
  # safely handle each part in a separate bucket to ensure proper interpretation (%s for
the text)
  # top line
  printf "%b%b%s%b%b\n" "$ESCon" "$UL_wall" "$_be_h_line" "$UR_wall" "$ESCoff"
  # content
  printf "%b%b %b%s%b %b%b\n" "$ESCon" "$V_wall" "$ESCoff" "$_be_text" "$ESCon" "$V_wall"
 "$ESCoff"
  # bottom
  printf "%b%b%s%b%b\n" "$ESCon" "$LL_wall" "$_be_h_line" "$LR_wall" "$ESCoff"
  unset -v _be_text _be_len _be_h_line
}

box_shift() # (POSIX) DEC box drawing using SO/SI toggling
{ # @usage box_shift "text"
  # @args $1: String to be boxed
  # @note Uses SO/SI toggling (faster)
  # @note text is displayed inside a single-wall box
  _bs_text="$1"
  _bs_len=${#_bs_text}   # length of text determines width of box
  # use fixed width printf to create proper length string of spaces | tr into wall segmen
t
  _bs_h_line=$(printf "%*s" "$((_bs_len + 2))" "" | tr ' ' "${H_wall}")
  # safely handle each part in a separate bucket to ensure proper interpretation (%s for
the text)
  # Top Line
  printf "%b%b%s%b%b\n" "$SO" "$UL_wall" "$_bs_h_line" "$UR_wall" "$SI"
  # Content
  printf "%b%b %b%s%b %b%b\n" "$SO" "$V_wall" "$SI" "$_bs_text" "$SO" "$V_wall" "$SI"
  # Bottom
  printf "%b%b%s%b%b\n" "$SO" "$LL_wall" "$_bs_h_line" "$LR_wall" "$SI"
  unset -v _bs_text _bs_len _bs_h_line
}

test_colors()      # (POSIX) print the joetoo RGB name of each eponymous color
{ _rgb_color_names="BLACK RED GREEN YELLOW BLUE MAG CYAN WHITE"
  _h=$(termheight); _w=$(termwidth); _current_row=$(( (_h / 2) - 4 )); _c_idx=0
  # ensure start row is within screen bounds
  if [ "$_current_row" -lt 1 ]; then _current_row=1; fi
  for _name in $_rgb_color_names; do
    _fg_code=$(( 30 + _c_idx ))
    _bg_code="${BG_DEFAULT}"
    # special case: if text is BLACK (30), set background to WHITE (47)
    if [ "$_fg_code" -eq "${BLACK}" ]; then _bg_code="${WBACK}"; fi;
    _esc_start=$(color "$_fg_code" 0 "$_bg_code")
    _colored_text="${_esc_start}${_name}${Boff}"
    # centering logic (use the raw name length)
    _col=$(( (_w / 2) - (${#_name} / 2) ))
    CUP "$_current_row" "$_col"
    printf '%b' "$_colored_text"
#    printf '%b' "$(CUP "$_current_row" "$_col") $_colored_text"
    _current_row=$(( _current_row + 1 ))
    _c_idx=$(( _c_idx + 1 ))
  done
  CUP "$_h" 1
#  printf '%b' "$(CUP "$_h" 1)"
  unset -v _rgb_color_names _h _current_row _c_idx _name _fg_code \
      _bg_code _esc_start _esc_reset _colored_text _w _col ; }
# @usage test_colors
# @note validates SGR color rendering and vertical centering logic
# @note special case: forces white background for BLACK text to ensure visibility
# @deps color, termheight, termwidth, CUP
```

```
test_terminal() # (POSIX} full-suite validation of cursor manipulation and terminal dimen
sions
{
    # clear screen and show dimensions
    CLR
    _W=$(termwidth)
    _H=$(termheight)
    # test absolute positioning (CUP) - print an X in each corner
    CUP 1 1; printf "X"
    CUP 1 "$_W"; printf "X"
    CUP "$_H" 1; printf "X"
    CUP "$_H" "$_W"; printf "X"
    # test centering
    _msg="Terminal: ${_W}x${_H}"
    _msg_len=${#_msg}
    _start_col=$(( (_W / 2) - (_msg_len / 2) )) # start message half-its-length left of c
enter
    _start_row=$(( (_H / 2) - 10 ))             # start far enough above middle to leave
room for colors
    CUP "$_start_row" "$_start_col"; SCP
    printf '%s' "_msg_len: $_msg_len"; RCP; CUD; SCP
    printf "%s\n" "$_msg"
    # run test_colors()
    test_colors
    # test save/restore cursor (SCP/RCP)
    CUP $((_start_row + 2)) "$_start_col"
    printf "Saving cursor position..."
    SCP
    # move away ...
    CUP 1 5 ; printf "[Moving]"
    # restore cursor to saved position and finish
    RCP; printf " ...Restored!"
    # test repeat() by drawing a screen-wide line of "=" chars
    CUP $((_start_row + 4)) 1
    repeat "=" "$_W"
    # move cursor to bottom for prompt
    CUP $(("$_H"-2)) 1; printf "\nTest Complete. Press Enter"
    read _junk < /dev/tty
    unset -v _W _H _msg _msg_len _start_col _start_row _junk
}
# @usage test_terminal
# @note tests: CLR (clear), CUP (positioning), SCP/RCP (save/restore), and repeat()
# @deps CLR, termwidth, termheight, CUP, SCP, RCP, repeat, test_colors

#=====[ Section 3: Core Validation Functions ]==========================================
====================

isint() # (POSIX) tests if $1 is a decimal integer
{ [ $# -eq 0 ] && return 1  # guard clause: check if an argument was provided
  _val=$1
  _test_val="${_val#-}"  # Remove leading dash (negative # ok; test the rest)
  case "$_test_val" in
    "" | *[!0-9]* ) unset -v _val _test_val; return 1 ;;
    *             ) unset -v _val _test_val; return 0 ;;
  esac ; }
# @usage isint "input"
# @args $1: the string to be validated as a decimal integer
# @ret 0 if valid integer; 1 if not
# @rule accepts leading '-' for negative values; rejects all non-digit characters
# @note mutually exclusive with isfloat_posix and ishexint in this suite
# @rule handles negative signs via parameter expansion ${1#-}
# @ex isint "-42"  # returns 0

ishexint() # (POSIX) recognizes hexadecimal integers per shell standards
{ # @usage ishexint "input"
  [ $# -eq 0 ] && return 1          # guard: return false if no argument is provided
  _val=$1;   _test_val="${_val#-}"  # parameter expansion: removes a leading dash if pres
ent
                                    # this allows the function to recognize negative hex
```

```
constants
  case "$_test_val" in
    0[xX][0-9a-fA-F]*)              # 1: Check for the required POSIX shell arithmetic pr
efix (0x or 0X)
                                    #    followed by at least one valid hex digit ([0-9a-
fA-F]*)
      _test_val="${_test_val#??}"   # +  Strip the prefix using parameter expansion
                                    #    #?? removes the first two characters (the '0x' o
r '0X')
      case "$_test_val" in          # +  Validate the remaining payload.
        "" | *[!0-9a-fA-F]*)        #    a. Reject if the payload is empty (just "0x") or
 contains non-hex characters
          unset -v _val _test_val; return 1
          ;;
        *)                          #    b. Success: the string is a valid hexadecimal co
nstant
          unset -v _val _test_val; return 0
          ;;
      esac ;;
    *)                              # 2: If it doesn't start with 0x/0X, it is not recogn
ized as hex in this scope
      unset -v _val _test_val
      return 1
      ;;
  esac
}
# @usage ishexint "input"
# @args $1: the string to be validated as a hexadecimal constant
# @ret 0 if valid hex (0x/0X); 1 if not
# @rule requires 0x or 0X prefix; handles optional leading '-'
# @note aligns with POSIX shell $((...)) arithmetic constant requirements
# @ex ishexint "0xAF12" # returns 0

isfloat_posix() # (POSIX) tests if $1 is a floating point decimal number
{ [ $# -eq 0 ] && return 1  # guard clause: check if an argument was provided
  _val=$1
  _test_val="${_val#-}"  # Remove leading dash (negative # ok; test the rest)
  # reject what we confirm in NOT float, of what is left accept only what has exactly one
 .; reject the rest
  case "$_test_val" in
    "" | "." | *[!0-9.]* ) unset -v _val _test_val; return 1 ;;    # reject if empty, jus
t a dot, or contains non-numeric/non-dot chars
    *"."*"."*            ) unset -v _val _test_val; return 1 ;;    # eject if there is mo
re than one dot
    *"."*                ) unset -v _val _test_val; return 0 ;;    # *success. must conta
in exactly one dot; no overlap with isint
    *                    ) unset -v _val _test_val; return 1 ;;
  esac
}
# @usage isfloat_posix "value"
# @args $1: value to test
# @ret 0 if valid float; 1 if not
# @rule requires exactly one decimal point; rejects scientific notation (e.g. 1e10)
# @rule fails on multiple dots or non-numeric characters
# @note aligns with bc "scale" semantics; distinct from isint
# @note does not support scientific notation (e.g. 1e10)

isnumeric() # (POSIX) tests if $1 is any valid numeric type (Dec, Hex, or Float)
{ isint "$1" || ishexint "$1" || isfloat_posix "$1" ; }
# @usage isnumeric "input"
# @args $1: the string to be tested
# @deps isint, ishexint, isfloat_posix
# @ret 0 if input matches any supported POSIX numeric type; 1 if not
# @note wrapper function for the strict POSIX validation suite

isnumber() # (POSIX) deprecated - use isint() or isnumeric(); isnumber tests if $1 is an
integer
{ # @usage isnumber "input"
  # @note maintained for legacy compatibility; redirects to universal isnumeric()
  isnumeric "$1"
```

```
}

isroot() # (POSIX) silent check for root UID
{ # @usage isroot || echo "not root"
  # @ret 0 if root; 1 if not
  # @note silent boolean check; does not emit messages or exit the shell
  [ "$(id -u)" -eq "${ROOT_UID:-0}" ]
}

checkroot() # (POSIX) run as root, of course (${var:-0} means default to 0)
{ if [ "$(id -u)" -ne "${ROOT_UID:-0}" ]; then E_message "${E_NOTROOT}"; echo; E_message
"exiting process [$$]"; exit 1; fi; }
# @usage checkroot
# @req root privileges
# @rule enforces root UID; exits process [$$] with status 1 on failure
# @ret 0 if root
# @deps E_message

checknotroot() # (POSIX) run as not root, of course (${var:-0} means default to 0)
{ if [ "$(id -u)" -eq "${ROOT_UID:-0}" ]; then E_message "${E_ROOT}"; echo; E_message "ex
iting process [$$]"; exit 1; fi; }
# @usage checknotroot
# @req non-root user account
# @rule prevents execution as root; exits process [$$] with status 1 on failure
# @ret 0 if not root
# @deps E_message

checkboot() # (POSIX) check fstab for boot/efi mountpoints and verify mounted status
{ _ck_fstab="/etc/fstab"; _ck_targets="/boot /efi /boot/efi"
  _ck_i=0; _ck_status=0
  for _ck_tgt in $_ck_targets; do
    # check if target is a defined (non-commented) mount point in fstab
    if grep -v "^${W0}#" "$_ck_fstab" | grep -q "${W1}${_ck_tgt}${W1}"; then
      # defined, so next verify it is currently mounted
      if ! findmnt -nlo TARGET,SOURCE "$_ck_tgt" >/dev/null 2>&1; then
        _ck_status=$(( _ck_status|(1<<_ck_i) ))   # set the ith most significant bit of s
tatus
      fi
    fi
    _ck_i=$((_ck_i + 1))
  done
  unset -v _ck_fstab _ck_targets _ck_tgt _ck_i
  return "$_ck_status"
}
# @usage checkboot
# @ret 0: all fstab-defined boot targets are mounted
# @ret bit 1 set: /boot should mounted but is not
# @ret bit 2 set: /efi should be mounted but is not
# @ret bit 3 set: /boot/efi should be mounted but is not
# @ex return 3 (binary 11) means both /boot /efi should be mounted, but neither is
# @deps grep findmnt
# @rule aggreagaate check for /boot, /efi, and /boot/efi

old_checkboot() # check to see if /boot is a mountpoint and is properly mounted
{ _fstab_check=$(grep -E "${W1}/boot(${W1}|$)" /etc/fstab | grep -v "^${W0}#") # find /bo
ot entries; strip comments
  if [ -z "$_fstab_check" ]; then set -- 2; # not supposed to be a mountpoint
  elif grep -qE "${W1}/boot(${W1}|$)" /proc/mounts 2>/dev/null || \
     grep -qE "${W1}/boot(${W1}|$)" /etc/mtab 2>/dev/null; then
    set -- 0; # properly mounted
  else
    set -- 1; # Listed in fstab but NOT mounted
  fi; unset -v _fstab_check; return ${1:-255}; }  # default 255 (so we know it broke if $
1 is empty)
# @usage checkboot
# @ret 0: mounted; 1: in fstab but unmounted; 2: no /boot entry in fstab
# @ret 255: internal logic failure
# @rule uses REGEX_POSIX (W0, W1) to parse /etc/fstab and /proc/mounts safely
# @note status 2 indicates /boot is likely just a directory on the root partition
# @note fstab format: |<device|uuid> /boot ...| i.e. mountpoint is colunm 2 preceded by <
```

```
device|uuid> and ${W1}
# @cont followed by whitespace or (rarely) simply the end of the line -- (${W1}|$)

checkshell() # reset colors if this shell is not interactive
{ [ -z "$PS1" ] && . ${non_interactive_header}; }
# @usage checkshell
# @req non_interactive_header variable must be defined in Section 1
# @rule sources a fallback header if PS1 is null (non-interactive shell)
# @note used to prevent ANSI escape "garbage" in cron jobs or piped output

validate_logfile()  # validate existence and ownership of logFile
{ d_log_message "in validate_logfile" 5
  _logFile_owner_uid=""; _logFile_group_gid=""
  _user_uid=""; _user_gid=""
  _msg=""
  log_message_n "looking for logFile [${logFile}]"
  if [ ! -f "${logFile}" ] ; then
      log_echo_e_n " (${BRon}not found${Boff})"
      log_right_status 1
      _msg="logFile [${logFile}] not found\n"
      _msg="${_msg}${BRon}Please run (as root) ${BGon}touch ${logFile}; chown ${user}:${u
ser} ${logFile}${Boff}"
      log_E_message "${_msg}"
      unset -v _logFile_owner_uid _logFile_group_gid _user_uid _user_gid _msg
      return 1
  else
      log_echo_e_n " (${BGon}found${Boff})"
      log_right_status $TRUE
      log_message_n "checking logFile ownership"
      _logFile_owner_uid=$(ls -n "${logFile}" | awk '{print $3}')
      _logFile_group_gid=$(ls -n "${logFile}" | awk '{print $4}')
      _user_uid=$(id -u "${user}")
      _user_gid=$(id -g "${user}")
      if [ "${_logFile_owner_uid}:${_logFile_group_gid}" = "${_user_uid}:${_user_gid}" ]
; then
          log_echo_e_n " (${BGon}${_logFile_owner_uid}:${_logFile_group_gid}${Boff})"
          log_right_status $TRUE
      else
          log_echo_e_n "(${BRon}${_logFile_owner_uid}:${_logFile_group_gid}${Boff})"
          right_status 1
          _msg="bad ownership on logFile [${logFile}]\n"
          _msg="${_msg}${BRon}Please run (as root) ${BGon}chown ${user}:${user} ${logFile
}${Boff}"
          log_E_message "${_msg}"
          unset -v _logFile_owner_uid _logFile_group_gid _user_uid _user_gid _msg
          return 1
      fi # ownership
  fi # existence
  unset -v _logFile_owner_uid _logFile_group_gid _user_uid _user_gid _msg
  return 0
}
# @usage validate_logfile
# @req logFile and user must be set apriori
# @ret 0: exists and owned by $user; 1: missing or bad ownership
# @deps log_message_n log_echo_e_n log_right_status log_E_message d_log_message
# @vars logFile (path); user (owner name)
# @note no arrays in ash/busybox, so don't use ${FUNCNAME[0]}
# @note in ash, the =+ operator works for arithmetic but not text concatenation
# @rule identifies owner/group UIDs via ls and awk for POSIX compatibility
# @rule emits colored "corrective" instructions to stderr on failure# @req logFile and us
er must be set apriori

#=====[ Section 4: String & Layout Utilities ]============================================
====================

_strip_ansi() # (POSIX) Strip ANSI escape sequences from stdin
{
  # Works with standard sed to remove color/UI codes for clean logging
  sed "s/${CSI}[0-9;]*[mK]//g"
}
```

```
# @usage _strip_ansi "string" OR echo "string" | _strip_ansi
# @args $*: the string containing ANSI sequences
# @ret String with all \033[...m sequences removed
# @note essential for calculating true "visible" string length
# @note regex pattern in "${CSI}[0-9;]*[mK]" is std for matching nearly all SGR (Select G
raphic Rendition) parameters
# @note global CSI=$(printf "${ESC}[") is defined at the top of this header

_translate_escapes() # (POSIX) translate layout control chars to spaces, preserve ESC
{
  # explicitly target "layout breakers", not the entire [:control:] class --
  # \010 = BS (backspace)
  # \011 = HT (tab)
  # \012 = LF (newline)
  # \013 = VT (vertical tab)
  # \014 = FF (form feed)
  # \015 = CR (carriage return)
  tr '\010\011\012\013\014\015' ' '
}
# @usage _translate_escapes
# @ret Cleaned string with layout-breakers converted to spaces
# @rule targets BS, HT, LF, VT, FF, CR (octal 010-015)
# @rule preserves ESC (octal 033) to maintain ANSI styling functionality

_get_msg_len() # helper not to be called directly (get actual printable length of message
)
{ # strip ANSI escape sequences, and use ${#var} to count what remains
  [ $# -lt 1 ] && return 1   # guard clause; reject if no message
  # normalize: translate layout breaking control chars single space
  # then strip ANSI to get the visible length
  _stripped=$(printf '%b' "$*" | _translate_escapes | _strip_ansi)
  _len="${#_stripped}"; echo "${_len:-0}" # if empty, return 0
  unset -v _stripped _len; }
# @usage _get_msg_len "string"
# @ret (int) count of visible characters
# @deps _translate_escapes _strip_ansi
# @rule uses translation strategy to ensure layout chars don't inflate count
# @rule uses "$*" to treat all arguments as a single message string and _strip_ansi to fi
lter it

get_longest() # echo length of longest string in input $@
{ _ans=0; for _x in "$@"; do _N=$(_get_msg_len "$_x"); [ $_N -gt $_ans ] && _ans=$_N; don
e ;
  printf '%s' "$_ans"; unset -v _ans _x _N; }
# @note switched from ${#_x} to _get_msg_len "$_x" to strip ansi;
# @note unlike _get_msg_len, which is a lookahead for word-wrapping decisions,
# @cont get_longest calculates the length of the longest element (like varnames for displ
ay_vaars)
# @cont but to ensure utility outside just single word entries in the column width need b
eing counted
# @cont it was switched from $@ (unquoted) to "$@"
#{ _ans=0; for _x in $@; do [ ${#_x} -gt $_ans ] && _ans=${#_x}; done ; printf '%s' "$_an
s"; unset -v _ans _x; }

repeat() # (POSIX) quickly output a string of char $1 (of len $2) (not for spaces)
{ _marker="${1:--}"; _reps="${2:-40}"; [ "${#_marker}" -gt 1 ] && \
  _marker=$(printf -- '%s' "${_marker%${_marker#?}}"); printf "%${_reps}s" | tr ' ' "$_ma
rker" ; }
# @usage repeat "marker" "count"
# @args $1 char to repeat; $2 number of times
# @ret (emits) string of repeated characters

pad() # generate pad for cur(str $1) + sep(str $2) w min(int $3) color(str $4) marker(cha
r $5); filling col(int $6)
{ # initial guard - all six arguments must exist (some can be null to be defaulted)
  [ $# -lt 6 ] && { E_message "#args: $#; 6 args required: cur, sep, min, color, marker,
col"; return 1; }
  # assign localized variables with parameters received or defaults where allowable (seco
ndary guard)
  _pad_cur="$1"; _pad_sep="${2:- }"; _pad_min="${3:-2}";
```

```
  _pad_color="$4"; _pad_marker="${5:-.}"; _pad_col="$6"
  # exercise additional guards --
    # $1: reject null current element
    [ -z "$_pad_cur" ] && { E_message "arg1 (str cur) cannot be null"; return 1; }
    # $2: remove ansi from marker (will be colored by $color if that is provided)
    _pad_sep=$(printf '%s' "$_pad_sep" | _translate_escapes | _strip_ansi)
    # $3: min is defaulted above :- (exists?) in "${3:-2}" ensures that null (or even uns
et) is reassigned
    # $4: initial guard ensures existence, and null is OK for printf,
    #     so, ensure color is a proper ANSI SGR sequence and not printable text (e.g.: ${
CSI}${RED};${BOLD}m = ${BRon})
      if [ -n "$_pad_color" ]; then
        case "$_pad_color" in
          "${CSI}"*m?*)    # reject if there is anything after the 'm'
            E_message "arg4 (color) contains trailing printable characters"; return 1 ;;
          "${CSI}"*m) : ;; # Valid sequence, do nothing and continue
          *)               # everything else is malformed or printable text
            E_message "arg4 (color) must be a valid ANSI sequence (e.g. \${CSI}31m)"; ret
urn 2 ;;
        esac
      fi
    # $5: marker is defaulted above :- (exists?) in "${5:-.}" ensures that null (or even
unset) is reassigned
    # $5: remove ansi from  separator (will be colored by $color if that is provided) and
 truncate
      _pad_marker=$(printf '%s' "$_pad_marker" | _translate_escapes | _strip_ansi)
      # truncate marker to a single character
      # ? matches a single char, so ${_pad_marker#? returns all after the first (result)
      # ${_pad_marker%  strips shortest match for (result) from the tail - leaving only t
he first
      _pad_marker="${_pad_marker%${_pad_marker#?}}"
    # $6: reject null column width
    ! isint "$_pad_col" && { E_message "arg6 (int col) cannot be null"; return 2; }
  # use _get_msg_len for ANSI stripping just to get the length of current element
  _pad_cur_len=$(_get_msg_len "$_pad_cur")
  _pad_sep_len="${#_pad_sep}"   # already ANSI stripped abpve
  # calculate the pad length
  # (L->R): min + col - (cur_len + sep_len)
  # (R->L): col - (cur_len + sep_len) + min
  _pad_length=$(( _pad_min + _pad_col - (_pad_cur_len + _pad_sep_len) ))
  # generate the full pad as a string
#  d_echo "_pad_cur: $_pad_cur" 4
#  d_echo "_pad_sep: $_pad_sep" 4
#  d_echo "_pad_min: _pad_min" 4
#  d_echo "_pad_color: ${_pad_color}this${Boff}" 4
#  d_echo "_pad_marker: $_pad_marker" 4
#  d_echo "__pad_col: $_pad_col" 4
#  d_echo "_pad_length: $_pad_length" 4
  _pad_string=$(repeat "$_pad_marker" "$_pad_length")
  # output the pad
  printf "%b%s%b" "$_pad_color" "$_pad_string" "$Boff"
  unset -v _pad_cur _pad_sep _pad_min _pad_color _pad_marker _pad_col
  unset -v _pad_cur_len _pad_sep_len _pad_length _pad_string
  return 0
echo "got this far"; return 1
}
# @usage pad "cur" "sep" "min" "color" "marker" "col"
# @req all 6 args must be supplied; see below how some '' vals can default
# @note where . str cur = the current element in fixed width column
# @note ....... str sep = internal separator (e.g. ': ')
# @note ....... int min = the minimum number of pad chars ...... (default 2)
# @note ..... str color = ANSI escape sequence (e.g. '\033[32m')
# @note ... char marker = the character w which to construct pad (default '.')
# @note ....... int col = the columns total fixed width
# @note marker string will be stripped of ANSI and truncated to 1 char
# @deps _translate_escapes, _strip_ansi
# @warn failure to properly quote all args may result in unexpected behavior

echo_n_long() # wordwrap string; $1 current col, $2 = indent, remainder = message words
{ # @warn need to incorporate _msg=$(printf '%b' "$*" | tr '\t\n' '  ') to filter/transla
```

```
te these special cases - but don't want to break it so haven't done so yet
#echo -e "args: $@"; message "args: $@"; d_message "args: $@" 1  ## debugging "residue"
#  logFile=/var/tmp/_enl_trace.log  # debugging log
  if [ $# -lt 2 ] ; then E_message "minimum 2 args: current_col, indent; remaining args =
 msg words"; return 1; fi
  if ! isint $1 ; then E_message "Error: arg \$1 must be numeric starting caracter positi
on"; return 1; fi
  _enl_current_col=$1                                                  # assign _enl_current_co
l
  if isint "$2"; then _enl_indent=$2; else _enl_indent=0; fi  # assign _enl_indent
  shift 2  # remainder of $* are the words of the message to be smart-wrapped
  # translate internal escapes ([:control:] chars like \n, \t, \r, etc.) to spaces
  # (do not _strip_ansi b/c want to preserve color codes, etc; only remove those in subsh
ell to _get_msg_len)
  _enl_msg=$(printf '%b' "$*" | _translate_escapes)   # operate on entire set of args as
a single string "$*"
  # perform controlled split on the remaining parameters ($@)
  set -f     # Disable globbing to prevent interpretation of "*" as wildcard to be expand
ed, etc.
  # safely convert a positional parameters (either multiple args or a single quoted strin
g) into individual words
  set -- $_enl_msg    # set with $_enl_msg unquoted so each split word becomes a position
al parameter
  set +f     # re-enable globbing (minimize the "blast radius" of the globbing-disabled s
tate)
  # after shift and reassembly; set a default message argument to "<empty>" if none provi
ded
  [ $# -eq 0 ] && set -- "$@" "<empty>"
  _enl_tw=$(termwidth); isint "$_enl_tw" || _enl_tw=80  # ensure _enl_tw is a save number
 for math
  # calculate the "right_status collision zone" so the last wrap keeps us clear of it
  _enl_status_width=6   # e.g. "[ Ok ]" = 6 char
  _enl_buffer=4         # padding for more visually appealing wrapping
  _enl_collision_limit=$(( _enl_tw - _enl_status_width - _enl_buffer ))
  [ "$_enl_collision_limit" -lt 0 ] && _enl_collision_limit=0  # clamp to avoid negative
math errors
  _enl_line_len="$_enl_current_col" # initialize with starting position
  _enl_trace=""  # for debuggin - use this to determine who printed what
  # smartly consider whether to print each word or wrap it to the _enl_indent
  _enl_FIRST_WORD=$TRUE  # don't wrap for first word
#  d_log_message "_enl_tw: $_enl_tw" 5
#  d_log_message "_enl_current_col: $_enl_current_col" 5
#  d_log_message "_enl_indent: $_enl_indent" 5
#  d_log_message "_enl_collision_limit: $_enl_collision_limit" 5
#  d_log_message "_enl_line_len: $_enl_line_len" 5
#  d_log_message "_enl_FIRST_WORD: $(TrueFalse $_enl_FIRST_WORD)" 5
  while [ $# -gt 0 ]; do
    # measure the length of remaining message
    _enl_total_tail_len=$(_get_msg_len "$*")
    _enl_x_peak=$1; _enl_x_peak_len=$(_get_msg_len "$_enl_x_peak") # preview; don't shift
 until necessary
#    d_log_message "   [in while] _enl_total_tail_len: $_enl_total_tail_len" 5
#    d_log_message "   [in while] _enl_x_peak: $_enl_x_peak" 5
#    d_log_message "   [in while] _enl_x_peak_len: $_enl_x_peak_len" 5
    # if not the first word check if the remaining message will cross the collision limit
...
    if [ $_enl_FIRST_WORD ]; then
        # start of line case (do we wrap, finish, or step?)
        if [ $(( _enl_line_len + _enl_x_peak_len )) -gt $_enl_collision_limit ]; then # w
ont fit; must wrap, but how?
            if [ "$_enl_line_len" -ne "$_enl_indent" ] ; then   # this is the very first
word (first line) - do  a soft wrap
                # very first word is too big, switch to midline logic to wrap (stay "gree
dy")
#                _enl_trace="${_enl_trace}<noprint-softwrap> W1\n"
#                _enl_trace="${_enl_trace}   _enl_line_len: $_enl_line_len\n"
#                _enl_trace="${_enl_trace}   _enl_collision_limit: $_enl_collision_limit\
n"
                printf "\n%b" "$(repeat ' ' $_enl_indent)"                  # W1: WRAP -
soft-wrap
```

```
                _enl_line_len=$_enl_indent                                  # Note: _enl_
FIRST_WORD remains true on new line
                continue   # skip the flow fi; fi; done; and while [] check; just restart
 while loop again with new state
            else  # this is first word of other than first line; non-fit is emergency ==>
 force hard wrap
                _enl_x="$1"; _enl_x_len=$(_get_msg_len "$_enl_x"); shift    # E2: WRAP -
hard-wrap
                printf '%b' "$_enl_x"                                       # print wheth
er it fits or not
#                _enl_trace="${_enl_trace}${_enl_x} E2\n"
                _enl_line_len=$(( _enl_line_len + _enl_x_len )); _enl_FIRST_WORD=$FALSE
            fi
        elif [ $(( _enl_line_len + _enl_total_tail_len )) -le $_enl_collision_limit ]; th
en
            printf '%b' "$*";                                               # F3: FINISH:
 it all fits, so print it all! (we're done)
#            _enl_trace="${_enl_trace}$* F3\n"
            break
        else  # start of line, first word exceeds limit
            _enl_x="$1"; _enl_x_len=$(_get_msg_len "$_enl_x"); shift        # S4: STEP: a
ssign word, measure, then drop from remaining message
            printf '%b' "$_enl_x"                                           #   print thi
s word (start of line)
#            _enl_trace="${_enl_trace}${_enl_x} S4\n"
            _enl_line_len=$(( _enl_line_len + _enl_x_len ))
            _enl_FIRST_WORD=$FALSE
        fi  # start-of-line finish or step
    else  # mid-line case (do we finish, step, or wrap?)
        if [ $(( _enl_line_len + _enl_total_tail_len + 1 )) -le $_enl_collision_limit ];
then
            printf " %b" "$*";                                              # F5: FINISH:
 it all fits, so print it all! (we're done)
#            _enl_trace="${_enl_trace}$* F5\n"
            break
        elif [ $(( _enl_line_len + _enl_x_peak_len + 1 )) -le $_enl_tw ]; then # remainde
r DOESN"T fit left of collision zone
            if  [ "$_enl_total_tail_len" -eq "$_enl_x_peak_len" ]; then     # this is the
 very last word
                # && [ $(( _enl_line_len + _enl_x_peak_len + 1 )) -gt "$_enl_collision_li
mit" ]
                #   is implied by the non selection of -le same things above for F5
                #fits in margin but last word (and we know WONT fit left of collision zon
e from above) - must wrap
                printf "\n$(repeat ' ' $_enl_indent)%s" "$_enl_x_peak"      # W6: WRAP/FI
NISH very last word, would fit in margin but not L of collision
                shift                                                       # drop for re
mainder (overkill - break will jump us out anyway)
#                _enl_trace="${_enl_trace}<last-word-soft-wrap> W6\n"
                _enl_line_len=$(( _enl_indent + _enl_x_peak_len ))          # also overki
ll - very last word already printed; don't need this any more
                break
            else
                # not last word - step (only last word needs to be bumped from collision
zone
                # so there is another line on which to put right_status
                _enl_x="$_enl_x_peak"; _enl_x_len="$_enl_x_peak_len"; shift # S7: STEP: a
ssign word, measure, then drop from remaining message
                printf " %b" "$_enl_x"                                      #   print thi
s word (it will fit)
#                _enl_trace="${_enl_trace}${_enl_x} S7\n"
                _enl_line_len=$(( _enl_line_len + _enl_x_len + 1 ))
                # _enl_FIRST_WORD remains $FALSE
            fi
        else
            printf "\n%b" "$(repeat ' ' $_enl_indent)"                      # W8: WRAP: w
ord wont fit, so print newline and indent
#            _enl_trace="${_enl_trace}<indent> W8\n"
            _enl_line_len=$_enl_indent; _enl_FIRST_WORD=$TRUE
        fi  # mid-line finish, step, or wrap
```

```
    fi  # first word?
  done
#  d_log_message "${_enl_trace}" 4
  # clean up variables to maintain "local" scope in sourced environments
  unset -v _enl_buffer _enl_collision_limit _enl_current_col _enl_FIRST_WORD
  unset -v _enl_indent _enl_line_len _enl_status_width _enl_total_tail_len
  unset -v _enl_trace _enl_tw _enl_x _enl_x_len _enl_x_peak _enl_x_peak_len
}
# @usage echo_n_long "$start" "$indent" "$message"
# @deps _translate_escapes, _strip_ansi
# @req The function must know two facts, for absolute reliability -- where it is starting
 and (b) how far to indent
# @rule $1 (start) must be supplied as a decimal integer. This informs the engine where t
he cursor is located before it starts printing. (putting it there is the call>
# @rule $2 (indent) must be set (either to a decimal integer or null value). This informs
 the engine where to put the indent on subsequent lines
# @note (1) This function is a fragment-aware smart-wrapping layout engine designed for p
osix ash compatibility
# @note (2) Primary objective: deterministic word-wrapping to protect the right_status co
llision zone
# @note (3) Secondary objective: visual alignment via hanging indentation
# @warn The engine is highly reliable. Onus is on the caller to assign and provide the co
rrect values for $1 and $2 and consequences of these choices are thus purely >
#
# Design and Maintenance Notes:
# _enl_current_col is then used to initialize tracking needed to calculate
#    (for every step) whether the considered output can fit on the current line or not,
#    thus enabling the FINISH/STEP/WRAP methodology (the basic algorithm of the function)
# _enl_indent indicates how far the engine should indent any second and/or subsequent lin
e(s)
#    and how to adjust its position-tracking when it word-wraps the output
#    If set to a null value, the function will assign _enl_indent=0 (no hanging indent)
# Guard/assignment logic rejects null input $1 (_enl_current_col),
#     to prevent non-deterministic wrapping behavior
# Guard/assignment logic allows a null input for $2 (_enl_indent),
#     in which case, it will assign _enl_indent=0 (default)
# use printf '%b' throughout, to handle ANSI color codes, etc
# loop uses set -- $* inside set -f/set +f globbing protection to safely force word-split
ting
#   of strings for word-by-word evaluation
# clean scope is maintained by unsetting all _enl_ prefixed variables at exit
# history: originally was a dedicated backend helper for display_vars, later generalized
#   to support multiple callers objectively
# 'set -- ...' (described above) tells the command to stop looking for options;
#   everything that follows is a positional argument (so e.g. "-o" or "--example"
#   won't be seen as command options
# The engine employs a word wrapping algorithm to display its message --
#   it begins wherever the cursor is when called, then -
#   it respects the callers indentation request for the left edge of every wrapped line,
and
#   it respects the space required for right_status on the right margin of the last line
#     of the message

alpha_words() # (POSIX) alphabetize the words in a quoted input string
{ [ $# -eq 0 ] && return 0  # guard clause; just return 0 success if no args
  for _x in "$@"; do printf "%s\n" "$_x"; done | LC_ALL=en_US.UTF-8 sort -db 2>/dev/null
|| \
    for _x in "$@"; do printf "%s\n" "$_x"; done | sort -db # Fallback if UTF-8 locale is
 missing
  unset -v _x ; }
# @usage alpha_words "word list"
# @args $* string of words to be sorted
# @ret newline-separated list of alphabetized words
# @rule uses LC_ALL=en_US.UTF-8 sort -db for dictionary-order sorting
# @note sort -d (dictionary): Ignores punctuation and only looks at letters, digits, and
blanks
# @note sort -b (ignore leading blanks) in case string has accidental leading spaces

right_most() # (POSIX) echo the right-most character in a string variable $1
{ _rm_str="$1"; [ -z "$_rm_str" ] && return 1; # guard clause; reject empty string
```

```
printf -- '%s' "${_rm_str#${_rm_str%?}}"     # -- prevents printf from treating leading "
-" as a flag
unset -v _rm_str; }                          # clean up local variable
# @usage right_most "string"
# @args $1 input string
# @ret the single final character of the input string
# @rule uses POSIX parameter expansion ${str#${str%?}} to extract character
# @note -- prevents printf from treating leading "-" as a flag
# Note: ${_rm_str%?} removes the last character from the string;
#   ${_rm_str#...} tells the shell to "look at the full string and remove the prefix that
 matches ${_rm_str%?}
#   (together, the only thing left is the last character)
# echo ${_rm_str:$((${#_rm_str}-1)):1};        # other way 1 (non-POSIX string slicing; s
lower external echo)
# echo ${_rm_str} | sed "s|${_rm_str:0:-1}||"  # other way 2 (non-POSIX string slicing; s
lower external sed)
# echo ${_rm_str} | rev | cut -c 1 ; }          # other way 3 (non-POSIX rev/cut -c and s
lower external cut/rev)

left_most() # (POSIX) echo the left-most character in a string variable $1
{ _lm_str="$1"; [ -z "$_lm_str" ] && return 1
  printf -- '%s' "${_lm_str%${_lm_str#?}}"
  unset -v _lm_str ; }
# @usage left_most "string"
# @args $1 input string
# @ret the single first character of the input string
# @rule uses POSIX parameter expansion ${str%${str#?}} to extract character

print_center() # (POSIX) print text $2 centered on row $1
{ _pc_row="$1"; _pc_text="$2"; _pc_w=$(termwidth)
  _pc_len=${#_pc_text}; _pc_col=$(( (_pc_w / 2) - (_pc_len / 2) ))
  [ "$_pc_col" -lt 1 ] && _pc_col=1  # Prevent negative/zero column
  CUP "$_pc_row" "$_pc_col" ; printf '%s' "$_pc_text" ;
  unset -v _pc_row _pc_text _pc_w _pc_len _pc_col ; }
# @usage print_center "row" "text"
# @args $1 row number; $2 text to center
# @deps termwidth CUP
# @rule calculates column based on termwidth and raw string length
# @note does not currently account for ANSI length; use for raw text

#=====[ Section 5: Logic & Conversion ]=============================================
====================

TrueFalse() # (POSIX) echo "True" or "False", depending on truth of arg $1
{ # (T/F Y/N up/down)(1=0=$TRUE [Green], ""=$FALSE [Red])
  # @usage TrueFalse "input"
  # @args $1 value to evaluate (y/n, t/f, u/d, h/l, 0)
  # @ret "True" (exit 0), "False" (exit 1), or "unset" (exit 2)
  # @rule evaluates 0 as True based on global TRUE=0 [cite: 419]
  # @note handles High/Low and Up/Down for hardware-style logic checks
  case $1 in
    [yY]*|[tT]*|[uU]*|[hH]|"$TRUE"|"0" )
       printf '%s' "True"; return 0 ;;
    [nN]*|[fF]*|[dD]*|[lL]|"$FALSE"    )
       printf '%s' "False"; return 1 ;;
    *) printf '%s' "unset"; return 2 ;;
  esac ; }
# Note: "0" is included as part of the "true" line because of the definition TRUE=0 atop
this header
#       "1" is NOT included in the "false" line - return 2 (unset)
#       this is a deliberate choice b/c binary numbering 1/0 is the opposite
#       High/Low and Up/Down are included instead

status_color() # (POSIX) emit escape seq for color per arg $1
{ # (T/F Y/N up/down)(1=0=$TRUE [Green], ""=$FALSE [Red])
  # @usage status_color "input"
  # @args $1 logic value to colorize
  # @ret emits $BGon (green), $BRon (red), or $BWon (white)
  # @rule maps 0 to green and 1 to red for standard exit code visualization
  case $1 in [yY]*|[tT]*|[uU]*|[hH]|"$TRUE"|"0"  ) printf '%b' ${BGon};;
```

```
           [nN]*|[fF]*|[dD]*|[lL]|"$FALSE"|"1" ) printf '%b' ${BRon};;
           *) printf '%b' ${BWon};; esac ; }
# Note: "0" is included as part of the "true" line because of the definition TRUE=0 atop
this header
#       "1" is NOT included in the "false" (though that would map to exit code use)
#       this is a deliberate choice b/c binary numbering 1/0 is the opposite
#       High/Low and Up/Down are included instead
# 20260119 - update trying 1 among map to BRon (test if this "breaks" something)

vercomp() # (almost POSIX)(portible w/ coreutils) compare versions return [0:equal|1:grea
terthan|2:lessthan]
{ # @usage vercomp "v1" "v2"
  # @args $1 candidate version; $2 comparison version
  # @ret 0:equal | 1:greater than | 2:less than
  # @deps sort -V (coreutils)
  # @rule uses sort -V for natural version sorting (e.g., 1.10 > 1.2)
  _candidate1="$1"; _candidate2="$2"; _first=""
  if [ "$_candidate1" = "$_candidate2" ]; then
    unset -v _candidate1 _candidate2 _first
    return 0;
  else
     _first=$(printf '%b' "$1\n$2" | sort -V | head -n1);
    if [ "$_first" = "$_candidate2" ]; then
      unset -v _candidate1 _candidate2 _first
      return 1;
    else
      unset -v _candidate1 _candidate2 _first
      return 2;
    fi;  # _first
  fi;  # _candidates
}

show_result() # (POSIX) show the result of vercomp comparison, in english
{ case $1 in
    0) printf "%b\n" "${BGon}=${Boff}"; return 0;;
    1) printf "%b\n" "${BCon}>${Boff}"; return 0;;
    2) printf "%b\n" "${BRon}<${Boff}"; return 0;;
    *) printf "%b\n" "${BYon}*** Error ***${Boff}"; return 1;;
  esac ; }
# @usage show_result "code"
# @args $1 integer code from vercomp (0, 1, or 2)
# @ret colorized mathematical operator string (=, >, <)

my_date() # (POSIX) echo date string (DDD MM dd hh:mm:ss TZ yyyy)
{ _date_string=$(date "+%a %b %d %H:%M:%S %Z %Y"); printf "%s\n" "$_date_string" ; unset
-v _date_string; }
# @usage my_date
# @ret string formatted as "Wed Jan 21 12:47:00 EST 2026"
# @note standard POSIX date formatting

my_short_date() # (POSIX) echo shorter date string (yyyymmdd-timehack)
{ _date_string=$(date "+%Y%m%d-%H:%M:%S"); printf "%s\n" "$_date_string" ; unset -v _date
_string; }
# @usage my_short_date
# @ret string formatted as "20260121-12:47:00"
# @note optimized for log file naming or compact timestamps

initialize_vars() # initialize values for vars in $@; incl bool. lv.
{ # @usage initialize_vars [type.var_name] ...
  # @args $@ list of variables to initialize, optionally prefixed by type (e.g., bool.VAR
)
  # @ret 0: success; 1: fail (no args); 2: fail (invalid type)
  # @deps d_message d_message_n d_right_status
  # @rule type prefixes: bool (set $FALSE), yn (set 'no'), lv (set '')
  # @rule handles untyped variables by initializing them to NULL ('')
  # @note uses eval for dynamic variable assignment; handles case-insensitive type prefix
es
  # @note skips 'verbosity' and 'VERBOSE' to prevent accidental reset of script state
  [ $# -lt 1 ] && {  unset -v _iv_item _iv_parmlist _iv_var_name _iv_type _iv_var; return
 1; }
```

```
  _iv_parmlist=""
  for _iv_item in "$@"; do case "$_iv_item" in
      *verbosity*|*bool.VERBOSE*|*VERBOSE*) continue ;;
      *) _iv_parmlist="$_iv_parmlist $_iv_item" ;;
  esac; done
  d_message "initializing vars: $_iv_parmlist" 5
  for _iv_var_name in $_iv_parmlist
  do
    case $_iv_var_name in
      *.* )   # if echo $_iv_var_name | grep -q "\."; then
        # initialize by type
        _iv_type="${_iv_var_name%.*}"   # strip off chars right of and .
        _iv_var="${_iv_var_name#*.}"    # strip off chars left of and .
        case ${_iv_type} in
          [bB]*  )
            # (set $FALSE) boolean convention: parameter is "bool.${_iv_var}" where ${_iv
_var} is e.g. "VERBOSE"
            d_message_n "  (${_iv_type}) initializing ${_iv_var} FALSE ..." 2
            eval "${_iv_var}=\"\$FALSE\"" ; d_right_status $? 2
          ;;
          [yY]*  )
            # (set "no") yes/no convention: parameter is "yn.${_iv_var}" where ${_iv_var}
 is e.g. "EIX"
            d_message_n "  (${_iv_type}) initializing ${_iv_var} (no) ..." 2
            eval "${_iv_var}='no'" ; d_right_status $? 2
          ;;
          [lL]*  )
            # (set "") lonstring convention: parameter is "lv.${_iv_var}" where ${_iv_var
} is e.g. "CMDLINE"
            d_message_n "  (${_iv_type}) initializing ${_iv_var} NULL ..." 2
            eval "${_iv_var}=''" ; d_right_status $? 2
          ;;
          *      )
            # invalid variable _iv_type
            eval "${_iv_var}=''" ; d_right_status $? 2
            unset -v _iv_item _iv_parmlist _iv_var_name _iv_type _iv_var
            return 2
          ;;
        esac  # type
        ;;
      "BREAK" )  # elif [ "$_iv_var_name" = "BREAK" ] ; then
        # ignore BREAK
        _iv_type=""
        d_message_n "  ignoring BREAK ..." 2
        d_right_status $TRUE 2
        ;;
      * )  # else
        # untyped - initialize null
        _iv_type=""
        _iv_var="$_iv_var_name"
        d_message_n "  (${_iv_type}) initializing ${_iv_var} NULL ..." 2
        eval "${_iv_var}=''" ; d_right_status $? 2
        ;;
    esac  # _iv_var_name (fi)
    d_message "about to initialize $_iv_var_name of _iv_type: [ $_iv_type ]" 4
  done
  unset -v _iv_item _iv_parmlist _iv_var_name _iv_type _iv_var
  return 0
}


display_vars()      # show vals for vars in $@; incl bool/lv/; $1=longest
{
  if ! isint $1 ; then
    E_message "Error: arg \$1 must be numeric longest expected input var name"
    exit
  fi
  if [ $# -le 1 ] ; then set $@ "<empty>"; fi  # insert placeholder "<empty>" as $2
#  logFile=/var/tmp/_dv_trace.log
  _dv_maxlen=$(($1 + 2))  # pad longest by 2 "." chars (others will have more dots)
```

```
  shift  # drop $1 (length of longest item to display in column)
  # loop through list of variable names and pad each accordingly
  for _dv_var_name in $@;  do
    _dv_type="${_dv_var_name%.*}"   # strip off chars right of and .
    _dv_var="${_dv_var_name#*.}"    # strip off chars left of and .
    _dv_pad_len=$(_get_msg_len "$_dv_var")  # strip ansi and then count length
    _dv_pad=" $(repeat '.' $(( _dv_maxlen - _dv_pad_len )) )"
    case "$_dv_var_name" in
      *.* )  # display by type
        case ${_dv_type} in
          [bB]*  )
            eval "_dv_key=\$${_dv_var}"
            message "${BCon}${_dv_var}${_dv_pad}: ${BBon}[ $(status_color ${_dv_key})$(Tr
ueFalse ${_dv_key}) ${BBon}]${Boff}"
            ;;
          [yY]*  )
            eval "_dv_key=\$${_dv_var}"
            message "${BCon}${_dv_var}${_dv_pad}: ${BBon}[ $(status_color ${_dv_key})${_d
v_key} ${BBon}]${Boff}"
            ;;
          [lL]*  )
            eval "_dv_key=\$${_dv_var}"
            # | * longestvarname ..: [
            # |123              4  567  (_dv_maxlen already padded for two dots)
            # [disregard - should work quoted now] use with ${_dv_key} (unquoted) so echo
_n_long can split and wrap words properly
            _dv_start_col=$(( _dv_maxlen + 7 )); _dv_indent="${_dv_start_col}"
#            d_log_message "_dv_var: $_dv_var" 5
#            d_log_message "_dv_maxlen: $_dv_maxlen" 5
#            d_log_message "_dv_start_col: $_dv_start_col" 5
#            d_log_message "_dv_indent: $_dv_indent" 5
            message_n "${BCon}${_dv_var}${_dv_pad}: ${BBon}[ ${Boff}"
            echo_n_long "${_dv_start_col}" "${_dv_indent}" "${_dv_key}"
            printf "%b\n" " ${BBon}]${Boff}"
            ;;
          *      )
            # invalid variable _dv_type
            E_message "  (${_dv_type}) invalid variable type [${_dv_type}] for variable [
${_dv_var_name}]"
            exit
            ;;
        esac
        ;;
      BREAK) printf "\n" ;; # BREAK, obviously (ignore it)
      * ) # untyped
        _dv_type=""
        _dv_var="$_dv_var_name"
        eval "_dv_key=\$${_dv_var}"
        _dv_pad=" $(repeat '.' $(( _dv_maxlen - _dv_pad_len )) )"
        message "${BCon}${_dv_var}${_dv_pad}: ${BBon}[ ${Boff}${_dv_key} ${BBon}]${Boff}"
        ;;
    esac
  done
  unset -v _dv_maxlen _dv_var_name _dv_type _dv_var _dv_pad _dv_key _dv_start_col _dv_ind
ent
  return 0
}
# @usage display_vars $longest "type.var1" "type.var2" ...
# @args $1 integer: length of the longest variable name for column alignment
# @args $2+ list of variable names (optionally typed: bool.VAR, yn.VAR, lv.VAR)
# @ret colorized table of variable states emitted to stdout
# @deps isint _get_msg_len repeat TrueFalse status_color message message_n echo_n_long
# @deps _translate_escapes, _strip_ansi
# @rule type bool: renders [ True/False ] based on global logic
# @rule type yn: renders [ yes/no ] as raw strings
# @rule type lv: renders [ value ] with word-wrap (echo_n_long) for long strings
# @note uses eval for dynamic symbol resolution; ensures layout-aware padding via repeat

show_config()       # show config with get_longest and display_vars
{ d_message "in show_config" 5 ; separator "${PN}-${BUILD}" "(configuration)"
```

```
  _longest=$(get_longest ${varlist}) ; display_vars ${_longest} ${varlist} ;
  unset -v _longest; }
# @usage show_config
# @vars varlist (list of variables to display)
# @vars PN, BUILD (for separator title)
# @deps d_message separator get_longest display_vars
# @rule automatically calculates column width via get_longest

#=====[ Section 6: Logging, Messaging, & UI ]=============================================
======================

#-----[ Baseline Messaging ]-------------------------------------------------------------
---------------------
separator()         # draw a horizontal line with preface $1 and title $2
{ # to facilitate separation of portions of the output of various scripts
  # @usage separator "preface" "title"
  # @args $1 preface string (defaults to hostname); $2 title string (defaults to script n
ame)
  # @deps termwidth repeat
  # @rule calculates terminal width to draw a full-width horizontal rule
  # @note subtracts 8 characters for brackets and spacing to prevent wrapping
  _s_preface=${1:-$(hostname)}; _s_title=${2:-$(basename "$0")}; _s_tw=$(termwidth)
  _s_reps=$(( _s_tw - ( ${#_s_preface} + ${#_s_title} + 8 ) ))
  [ "$_s_reps" -lt 0 ] && _s_reps=0  # clamp to 0 if negative
  printf '%b' "${BYon}---[${BRon} ${_s_preface} ${BCon}${_s_title} ${BYon}]"
  repeat '-' "$_s_reps"; printf "%b\n" "${Boff}"
  unset -v _s_preface _s_title _s_tw _s_reps; }

non_stty_separator()       # (deprecated; use separator)
{ E_message "${BYon}Warning:${Boff} non_stty_separator is deprecated (use separator(), wh
ich has been upgraded)"
  separator "$@"; }
# @usage non_stty_separator "preface" "title"
# @note legacy wrapper; redirects to separator() after emitting a warning

prompt()            # (POSIX) set external variable $answer based on reponse to prompt $1
{ _ps="$1"; answer=""; printf '%b' "\n\n"; CUU; SCP;
while ! expr "$answer" : '[yYnN]' >/dev/null; do
RCP; printf '%b' "$(repeat ' ' $(termwidth))";  ## blank the line
RCP; message_n "$_ps [Y/n]: " && read -r answer < /dev/tty; done; unset -v _ps ; }
# @usage prompt "question text"
# @args $1 string: the question to ask the user
# @vars answer (global): modified by this function
# @ret exit 0 on valid [y/n] input
# @deps CUU SCP RCP termwidth repeat message_n
# @rule loops until a valid y/n response is received from /dev/tty
# @note uses terminal escape codes to blank the line during redraw

new_prompt()        # (POSIX) set external variable $answer based on reponse to prompt $1
{ _ps="$1"; response=""; printf '%b' "\n\n"; CUU; SCP;
while ! expr "$response" : '[yYnNsS]' >/dev/null; do
RCP; printf '%b' "$(repeat ' ' $(termwidth))";  ## blank the line
RCP; message_n "$_ps [Yes/no/skip|Yns]: " && read -r response < /dev/tty; done; unset -v
_ps ; }
# @usage new_prompt "question text"
# @args $1 string: the question to ask the user
# @vars response (global): modified by this function
# @ret exit 0 on valid [y/n/s] input
# @deps CUU SCP RCP termwidth repeat message_n
# @rule loops until a valid Yes/No/Skip response is received from /dev/tty

confirm_continue_or_exit() # prompt() for confirm to continue (answer must be set globall
y)
{ answer="" ; _cce_msg="${BYon}Do you want to continue?" ; prompt "${_cce_msg}"
    case ${answer} in
        [yY]* ) message "Affirmative response from user; continuing" ;;
        [nN]* ) E_message "Negative response from user; quitting";
                # safeguard: only exit if running as a script, otherwise return 1
                case "$-" in
                    *i*) return 1 ;; # interactive: just return failure
```

```
                    *)   exit 1 ;; # script: kill execution
                esac ;;
        * ) E_message "invalid response to confirm_continue_or_exit"; return 1 ;;
    esac ; unset -v _cce_msg; return 0 ; }
# @usage confirm_continue_or_exit
# @vars answer (global): modified by internal call to prompt()
# @deps prompt message E_message
# @rule interactive shell: returns 1 on "no"; script: exits 1 on "no"
# @note provides a safety gate to prevent accidental batch execution

die()                        # display reason with E_message and exit with status 1
{ _die_msg="${1:-"unknown fatal error"}" ; E_message "${BRon}FATAL: ${_die_msg}${Boff}"
  unset -v _die_msg
  case "$-" in       # use the $- (current shell flags/options)
    *i*) return 1 ;; # interactive: return failure to terminal
    *)   exit 1   ;; # script: exit with error code
  esac; }
# @usage die "error message"
# @args $1 string: the reason for the fatal error
# @ret exit 1 (script) or return 1 (interactive)
# @deps E_message
# @rule detects shell state via $- to avoid closing interactive terminals

progress_inline() # show progress of silent process on next line $1=step, $2=total
{ _p_step=$1; _p_total=$2; _p_tw=$(termwidth); isint "$_p_tw" || _p_tw=80
  _p_percent=$(( (100 * _p_step) / _p_total ))
  # use 1/4 of screen width for the bar to keep room for text
  _p_bar_max=$(( _p_tw / 4 ))
  _p_filled=$(( (_p_bar_max * _p_step) / _p_total ))
  _p_empty=$(( _p_bar_max - _p_filled ))
  # determine which color to use for the "filled" markers and text NN% marker (heat map s
tyle)
  if [ "$_p_percent" -ge 90 ]; then _p_color="${BGon}"    # hot-good
  elif [ "$_p_percent" -ge 80 ]; then _p_color="${Gon}"   # warm-ok
  elif [ "$_p_percent" -ge 50 ]; then _p_color="${Con}"  # luke-warm
  elif [ "$_p_percent" -ge 25 ]; then _p_color="${Yon}"  # luke-warm
  else _p_color="${Ron}"; fi                              # cold

  # print inline with console output by using carriage return \r
  # assuming the process is silent, this stays on one line until the task is done
  # BUCKETS: - 1:markers_done 2:markers_not_done 3:color 4: xx% 5:reset 6:step 7:total
  # this printf command is going to be over 190 characters, and I don't like using "\"
  # to break commands over multiple lines because you cannot safely copy/paste those line
s
  # (e.g. for testing in another terminal), so I will use set to build it up
  #   start with the format string (see BUCKETS above note %b vs %s for ANSI color etc)
  set -- "\r Working: [%b%b] %b%3d%%%b (%d/%d)"
  #   append the bar segment buckets
#  set -- "$@" "${_p_color}$(repeat "#" $_p_filled)${Boff}"    # colored progress bar
  set -- "$@" "${Bon}$(repeat "#" $_p_filled)${Boff}"         # monotone progress bar
#  set -- "$@" "$(repeat "#" $_p_filled)${Boff}"               # uncolored monotone progr
ess bar
  set -- "$@" "${BBon}$(repeat "." $_p_empty)${Boff}"
  #   append the remaining status variables
  set -- "$@" "${_p_color}" "$_p_percent" "${Boff}" "$_p_step" "$_p_total"
  #   execute (double quotes are vital here, to preserve the quoting of the arguments)
  printf "$@"
  # on 100%, move to next line so the bar remains as a record
  [ "$_p_step" -eq "$_p_total" ] && printf "\n"
  unset -v _p_step _p_total _p_tw _p_percent _p_bar_max _p_filled _p_empty _p_color ; }
# @usage progress_inline $current $total
# @args $1 current step; $2 total steps
# @deps termwidth isint repeat
# @rule renders a horizontal progress bar using 1/4 of terminal width
# @rule implements a heat-map color scale for the percentage text
# @note uses "set --" to build complex printf commands without line continuations
# @note automatically emits newline upon reaching 100% to preserve the bar

sh_countdown()      # count-down seconds (arg $1)
{ # special note: bb sh cannot use base#number notation with prefix 10# to ensure interpr
```

```
etation as base 10
  _shc_rem_time="${1:-30}"  # guard; default to 30 sec. if no arg given
  [ "$1" = "" ] && message "No argument given, defaulting to 30 seconds"
  # execute countdown
  while [ "${_shc_rem_time}" -ge "0" ] ; do  # use -ge 0 to display the final state
    # bucket 1 - includes CSI + text + CSI
    # bucket 2 - the 2-digit integer formatted by %02d
    # bucket 3 - includes CSI + text + eraser spaces
    printf "\r%b%02d%b" \
      "${BGon}*${Boff} Pausing. [ ${BGon}" "$_shc_rem_time" "${Boff} ] seconds remaining.
..   "
    # exit loop immediatly after reaching/printing 00 (don't do "last" sleep cycle)
    [ "$_shc_rem_time" -eq 0 ] && break
    sleep 1; _shc_rem_time=$(($_shc_rem_time - 1))
  done
  printf "\n"  #clean up the line for the next command
  unset -v _shc_rem_time ; return 0 ; }
# @usage sh_countdown [seconds]
# @args $1 number of seconds to pause (defaults to 30)
# @ret 0 always
# @rule uses \r for in-place counter updates
# @note special note: bb sh cannot use base#number notation (don't prefix w 10#)
# @note breaks immediately at 0 to avoid an unnecessary final sleep cycle

message() # (POSIX) display text message $@
{ printf " ${BGon}*${Boff} " ; printf "$*"; printf "\n" ; }
# @usage message "msg string"
# @args $* the message to display (quoted string recommended)
# @ret formatted message with green asterisk prefix to stdout
# @note use "$*" to bind message as a single string (no word-split)

message_n()   # (POSIX) display text message $@ w/o newline
{ _mn_msg="$*"; _mn_msg_len=$(_get_msg_len "$_mn_msg")
  _mn_status_width=6   # e.g. "[ Ok ]" = 6 char
  _mn_buffer=4         # padding for more visually appealing wrapping
  # | * message
  #  123
  _mn_col=3; _mn_indent=3; _mn_tw=$(termwidth)
  _mn_collision_threshold=$(( _mn_tw - _mn_status_width - _mn_buffer - _mn_col )) # prep
to protect long msg
  [ "$_mn_collision_threshold" -lt 0 ] && _mn_collision_threshold=0   # clamp to 0 if neg
ative
  printf " ${BGon}*${Boff} ";  # print the "prefix *"
  if [ $_mn_msg_len -gt $_mn_collision_threshold ]; then
    echo_n_long "${_mn_col}" "${_mn_indent}" "$_mn_msg"  # use $_mn_msg unquoted for smar
t word-wrap; room for right_status; -5 to align w indent
  else
    printf '%s' "$*"        # use "$*" quoted to pass a single string
  fi; unset -v _mn_msg _mn_msg_len _mn_status_width _mn_buffer _mn_tw _mn_collision_thres
hold _mn_col _mn_indent; }
# @usage message_n "msg string"
# @args $* the message to display
# @deps _get_msg_len termwidth echo_n_long
# @rule uses a collision threshold to trigger echo_n_long for word-wrapping
# @note protects the right_status collision zone for subsequent right_status calls
# @note message family indent is 3; echo_n_long requires valid numbers for col and indent

W_message()   # (POSIX) display text warning message $@
{ printf " ${BYon}*${Boff} " ; printf "$@\n" ; }
# @usage W_message "warning string"
# @args $@ message components
# @ret formatted warning with yellow asterisk prefix to stdout

E_message()   # (POSIX) display text error message $@
{ { printf " ${BRon}*${Boff} "; printf "%b\n" "$*"; } >&2; }
# @usage E_message "error string"
# @args $* the error message
# @ret formatted error with red asterisk prefix to stderr (FD 2)

E_message_n()       # (POSIX) display text error message $@ (no CR)
```

```
{ { printf " ${BRon}*${Boff} " ; printf '%b' "$*" ; } >&2; }
# @usage E_message_n "error string"
# @args $* the error message
# @ret formatted error with red asterisk prefix to stderr without trailing newline
# @note protects the right_status collision zone for subsequent right_status calls

handle_result() # output ok_/_err_msg and right_stats to stdout
{ _exit_status="$1"; _ok_msg_in="$2"; _err_msg_in="$3"
  if [ "$_ok_msg_in" = "${no_msg}" ] ; then _ok_msg=""; else _ok_msg=" (${BGon}${2:-succe
ss!}${Boff})"; fi
  if [ "$_err_msg_in" = "${no_msg}" ] ; then _err_msg=""; else _err_msg=" (${BRon}${3:-er
ror!}${Boff})"; fi
  if [ "$_exit_status" -eq 0 ] ; then
      printf '%b' "${_ok_msg}"  # replicates echo -e -n
      right_status $TRUE
  else
      printf '%b' "${_err_msg}"  # replicates echo -e -n
      right_status 1
      unset -v _exit_status _ok_msg _err_msg _ok_msg_in _err_msg_in
      return 1
  fi
  unset -v _exit_status _ok_msg _err_msg _ok_msg_in _err_msg_in
  return 0; }
# @usage handle_result "$?" "ok_msg" "err_msg"
# @args $1 exit status; $2 success message; $3 error message
# @deps right_status status_color
# @rule handles optional message suppression via no_msg variable
# @ret exit 0 on success; exit 1 on failure
# @note usage - message_n ...; <command>] ; handle_result "$?" "_ok_msg" "_err_msg"
# @note notes - substitute no_msg=" \b" for either, to "suppress"


right_status()      # output 8 char [ ok/fail ] status at the right margin
{ # @usage right_status "status"
  # @args $1 integer status code
  # @deps termwidth isint CUF
  # @rule calculates movement value as termwidth - 7 to protect right margin
  # @rule clamps move value to 0 to prevent negative CUF arguments
  # @ret exit 0 on success; exit 1 on movement failure
  # @ex message_n "good_test" ; right_status $?
  # @ex E_message_n "fail_test" && cat unexistent 2>/dev/null; right_status $?
  # @note go to start of line, *then* move fwd to 7 chars from the right margin
  # @note "[ xx ]" is 6 chars, so this should ensure no accidental extra line wrap # @usa
ge right_status "status"
  # @args status must be integer status code
  # guard - reject zero args
  # @ex message_n "good_test" ; right_status $?
  # @ex E_message_n "fail_test" && cat unexistent 2>/dev/null; right_status $?
  [ $# -lt 1 ] && { E_message "Error: 0 args. right_status requires integer status code";
 return 1; }
  _rs_status=$1; _rs_msg=""
  if [ "$_rs_status" -eq 0 ] ; then
    _rs_msg="${BBon}[ ${BGon}Ok ${BBon}]${Boff}"
  else
    _rs_msg="${BBon}[ ${BRon}!! ${BBon}]${Boff}"
  fi
  # go to start of line, *then* move fwd to 7 chars from the right margin
  # Note: "[ xx ]" is 6 chars, so this should ensure no accidental extra line wrap
  _rs_tw=$(termwidth)
  # ensure we have a valid width for arithmetic
  isint "$_rs_tw" || _rs_tw=80
  # calculate movement value separately for clarity/safety
  _rs_move=$(( _rs_tw - 7 ))
  # clamp move value to 0 to prevent negative arguments to CUF
  [ "$_rs_move" -lt 0 ] && _rs_move=0
  # move
  printf '%b' "\r"  # beginning of current line (return 1 if failed)
  if ! CUF "$_rs_move" ; then unset -v _rs_status _rs_msg _rs_tw _rs_move; return 1; fi
  # final status output
  printf '%b' "${_rs_msg}\n"; unset -v _rs_status _rs_msg _rs_tw _rs_move; return 0; }
```

```
non_stty_right_status()    # (deprecated; use right_status)
{
  E_message "${BYon}Warning:${Boff} non_stty_right_status is deprecated (use right_status
(), which has been upgraded)"
  right_status "$@"
}

#-----[ Logging ]-------------------------------------------------------------------
--------------

timestamp()         # (POSIX) echo a simple timestamp (yyyymmdd-hh:mm:ss)
{ printf "%s\n" "$(date +'%Y%m%d-%H:%M:%S')"; }
# @usage timestamp
# @ret string: formatted date/time (e.g., 20260121-13:45:00)
# @note used for unique filenames, log entries, and non-blocking timehacks

log_separator()     # (POSIX) output separator() to stdout and similar to logFile
{ _ls_preface=${1:-$(hostname)}; _ls_title=${2:-$(basename "$0")}
  _ls_ts="$(timestamp)"; _ls_tw=80  # standard fixed width for log files
  # construct the header string: "timestamp - ---[ preface title ]"
  _ls_clean_header="${_ls_ts} - ---[ ${_ls_preface} ${_ls_title} ]"
  _ls_reps=$(( _ls_tw - ${#_ls_clean_header} ))  # calculate the length of the rest of th
e  ----- line
  [ "$_ls_reps" -lt 0 ] && _ls_reps=0      # if negative, clamp to 0
  # output (console screen first, then log)
  separator "$_ls_preface" "$_ls_title"    # out to console screen
  # output a "clean" version (w/o colors) to the log file
  { printf '%s' "$_ls_clean_header"
    repeat '-' "$_ls_reps"
    printf "\n"
  } >> "$logFile"
  unset -v _ls_preface _ls_title _ls_ts _ls_clean_header _ls_reps ; }
# @usage log_separator "preface" "title"
# @args $1 preface; $2 title
# @vars logFile
# @deps timestamp separator repeat
# @rule emits colorized line to stdout and a 80-char timestamped plain-text line to $logF
ile
# @note enforces a standard 80-character fixed width for log files to ensure readability

log_message()       # (POSIX) message() to stdout and $logFile owned by $user
{ _ts="$(timestamp)"; message "$_ts - $*";
  # Filter the message through _strip_ansi before logging
  _clean_msg=$(printf '%s' "$*" | _translate_escapes | _strip_ansi)
  printf "%s - %b\n" "$_ts" "$_clean_msg" >> "$logFile"
   unset -v _ts _clean_msg; }
# @usage log_message "msg string"
# @args $* the message to log (treated as a single string)
# @vars logFile
# @deps timestamp message _translate_escapes _strip_ansi
# @rule uses "$*" to bind all arguments and filters them through the translation strategy
# @note dual-output: terminal receives timestamped color message; log receives plain text

log_message_n()       # (POSIX) message() to stdout and $logFile owned by $user
{ _ts="$(timestamp)"; message_n "$_ts - $*";
  # Filter the message through _strip_ansi before logging
  _clean_msg=$(printf '%s' "$*" | _translate_escapes | _strip_ansi)
  printf "%s - %b" "$_ts" "$_clean_msg" >> "$logFile"
   unset -v _ts _clean_msg; }
# @usage log_message_n "msg string"
# @args $* the message to log
# @vars logFile
# @deps timestamp message_n _translate_escapes _strip_ansi
# @rule suppresses newline in both stdout and $logFile
# @note protects the right_status collision zone for subsequent right_status calls
# @note useful for logging status start-points before a process completes
# @rule uses "$*" to treat all arguments as a single message string and _strip_ansi to fi
lter it
```

```
log_echo()          # (POSIX) echo to stdout and $logFile owned by $user
{ _ts="$(timestamp)"; printf "%s - %s\n" "$_ts" "$*" # screen gets color codes
  _clean_msg=$(printf '%s' "$*" | _translate_escapes | _strip_ansi)
  printf "%s - %s\n" "$_ts" "$_clean_msg" >> "$logFile"
  unset -v _ts _clean_msg; }
# @usage log_echo "msg string"
# @args $* the message string
# @vars logFile
# @deps timestamp _translate_escapes _strip_ansi
# @rule simple echo wrapper with automated timestamping and log archival
# @rule uses "$*" to treat all arguments as a single message string and _strip_ansi to fi
lter it

log_echo_e()        # (POSIX) echo -e to stdout and $logFile owned by $user
{ _ts="$(timestamp)"; printf "%s - %b\n" "$_ts" "$*"
  _clean_msg=$(printf '%s' "$*" | _translate_escapes | _strip_ansi)
  printf "%s - %b\n" "$_ts" "$_clean_msg" >> "$logFile"
  unset -v _ts _clean_msg; }
# @usage log_echo_e "msg string"
# @args $* the message containing escape sequences
# @deps timestamp _translate_escapes _strip_ansi
# @rule interprets backslash escapes (%b) for both terminal and log file
# @rule uses "$*" to treat all arguments as a single message string and _strip_ansi to fi
lter it

log_echo_n()        # (POSIX) echo -n "$1" to stdout and $logFile owned by $user
{ _ts="$(timestamp)"; printf "%s - %s" "$_ts" "$*"
  _clean_msg=$(printf '%s' "$*" | _translate_escapes | _strip_ansi)
  printf "%s - %s" "$_ts" "$_clean_msg" >> "$logFile"
  unset -v _ts _clean_msg; }
# @usage log_echo_n "msg string"
# @args $* the message string
# @deps timestamp _translate_escapes _strip_ansi
# @rule outputs string without newline; sanitizes for log file storage
# @rule uses "$*" to treat all arguments as a single message string and _strip_ansi to fi
lter it

log_echo_e_n()      # (POSIX) echo -e -n "$1" to stdout and $logFile owned by $user
{ _ts="$(timestamp)"; printf "%s - %b" "$_ts" "$*"
  _clean_msg=$(printf '%s' "$*" | _translate_escapes | _strip_ansi)
  printf "%s - %b" "$_ts" "$_clean_msg" >> "$logFile"
  unset -v _ts _clean_msg; }
# @usage log_echo_e_n "msg string"
# @args $* string with escape sequences
# @deps timestamp _translate_escapes _strip_ansi
# @rule interprets escapes via %b and suppresses trailing newline in both outputs
# @rule uses "$*" to treat all arguments as a single message string and _strip_ansi to fi
lter it

log_E_message()     # (POSIX) E_message() to stdout and $logFile owned by $user (like ech
o -e)
{ _ts="$(timestamp)"; E_message "$_ts - $*"
  _clean_msg=$(printf '%s' "$*" | _translate_escapes | _strip_ansi)
  printf "%s - %b\n" "$_ts" "$_clean_msg" >> "$logFile"
  unset -v _ts _clean_msg; }
# @usage log_E_message "error string"
# @args $* the error message components
# @vars logFile
# @deps timestamp E_message _translate_escapes _strip_ansi
# @rule directs colorized output to stderr and sanitized plain-text to $logFile
# @note dual-stream logic: terminal sees RED alert; log records timestamped event
# @rule uses "$*" to treat all arguments as a single message string and _strip_ansi to fi
lter it

log_E_message_n()   # (POSIX) E_message_n() to stdout and $logFile owned by $user (like e
cho -e)
{ _ts="$(timestamp)"; E_message_n "$_ts - $*"
  _clean_msg=$(printf '%s' "$*" | _translate_escapes | _strip_ansi)
  printf "%s - %b" "$_ts" "$_clean_msg" >> "$logFile"
  unset -v _ts _clean_msg; }
```

```
# @usage log_E_message_n "error string"
# @args $* the error message components
# @deps timestamp E_message_n _translate_escapes _strip_ansi
# @rule directs output to stderr and $logFile without trailing newline
# @note protects the right_status collision zone for subsequent right_status calls
# @note useful for logging error prefixes before a long-running diagnostic starts
# @rule uses "$*" to treat all arguments as a single message string and _strip_ansi to fi
lter it

loggit()            # (POSIX) send a job completion status message to the cron logger
{ [ $# -gt 2 ] && { E_message "invalid loggit arguments" ; return 1; }
  _loggit_exit_status="${2:-0}"
  _loggit_program="${1:-Unspecified}"
  case "$_loggit_exit_status" in
    0 ) _loggit_priority="notice"; _loggit_message="succeeded.";;
    * ) _loggit_priority="err"; _loggit_message="failed with exit status ${_loggit_exit_s
tatus}.";;
  esac
  logger -p "cron.$_loggit_priority" "$_loggit_program $_loggit_message"
  unset -v _loggit_exit_status _loggit_program _loggit_priority _loggit_message
}
# @usage loggit "program_name" [exit_status]
# @args $1 string: program name; $2 int: exit status (defaults to 0)
# @deps E_message logger
# @rule maps 0 to 'notice' and non-zero to 'err' for system logging
# @ret exit 0 on success; 1 on invalid arguments
# @note interfaces with the system logger (syslog) under the 'cron' facility

log_handle_result() # output ok_/_err_msg and right_stats to stdout and logFile
{ _exit_status="$1"; _ok_msg_in="$2"; _err_msg_in="$3"
  if [ "$_ok_msg_in" = "${no_msg}" ] ; then _ok_msg=""; else _ok_msg=" (${BGon}${2:-succe
ss!}${Boff})"; fi
  if [ "$_err_msg_in" = "${no_msg}" ] ; then _err_msg=""; else _err_msg=" (${BRon}${3:-er
ror!}${Boff})"; fi
  if [ "$_exit_status" -eq 0 ] ; then
      log_echo_e_n "${_ok_msg}"
      log_right_status $TRUE
  else
      log_echo_e_n "${_err_msg}"
      log_right_status 1
      unset -v _exit_status _ok_msg _err_msg _ok_msg_in _err_msg_in
      return 1
  fi
  unset -v _exit_status _ok_msg _err_msg _ok_msg_in _err_msg_in
  return 0; }
# @usage log_handle_result "$?" "ok_msg" "err_msg"
# @args $1 exit status; $2 success string; $3 error string
# @deps log_echo_e_n log_right_status
# @rule handles optional message suppression via no_msg variable
# @ret exit 0 on success; exit 1 on failure
# @ex log_message_n ...; <command>] ; log_handle_result "$?" "_ok_msg" "_err_msg"
# @note substitute no_msg=" \b" for either, to "suppress"

log_right_status()  # output rigit_status to stdout and logFile
{ # @usage log_right_status "status"
  # @args status must be integer status code
  # guard - reject zero args
  # @ex log_message_n "good_test" ; log_right_status $?
  # @ex log_E_message_n "fail_test" && cat unexistent 2>/dev/null; log_right_status $?
  [ $# -lt 1 ] && { E_message "Error: 0 args. right_status requires integer status code";
 return 1; }
  _lrs_status=$1; if [ "$_lrs_status" -eq 0 ] ; then
    _lrs_msg="${BBon}[ ${BGon}Ok ${BBon}]${Boff}"
    _lrs_plain="[ Ok ]"
  else
    _lrs_msg="${BBon}[ ${BRon}!! ${BBon}]${Boff}"
    _lrs_plain="[ !! ]"
  fi
  # get width for screen positioning
  _lrs_tw=$(termwidth); isint "$_lrs_tw" || _lrs_tw=80  # fallback to standard 80 cols
```

```
  printf "\r"
  if CUF $(( "$_lrs_tw" - 7 )) ; then printf "%b\n" "$_lrs_msg"; else printf " %b\n" "$_l
rs_msg"; fi
  # clean logging
  if [ -n "$logFile" ]; then
    printf "%s\n" "$_lrs_plain" >> "$logFile" || \
      { unset -v _lrs_msg _lrs_status _lrs_plain _lrs_tw; return 1 ; }
  fi
  unset -v _lrs_msg _lrs_status _lrs_plain _lrs_tw; return 0 ; }
# @usage log_right_status "status"
# @args $1 integer status code
# @deps termwidth isint CUF E_message
# @rule uses \r and CUF to align terminal status to the far right margin
# @rule provides a clean [ Ok ] or [ !! ] entry to $logFile without timestamping
# @note maintainer note: deliberate decision to NOT have log_right_status timestamp
# @note its entries to help this "context aware" logging scheme be more readable# maintai
ner note: deliberate decision to NOT have log_right_status timestamp
#   its entries to help this "context aware" logging scheme be more readable
#   the alternative would be to "revert" to an "atomic" logging scheme
#   in which logged action goes on a separate line with its own timestamp
# note: 7 is the offset from right margin for six char status
#  [ ok ] or [ !! ], because it leaves a one-char buffer to guarantee
#  exactly one (and no accidental extra) newline

log_show_config()   # show & log config w get_longest and display_vars
{ d_log_message "in lg_show_config" 5 ; log_separator "${PN}-${BUILD}" "(configuration)"
  _longest=$(get_longest ${varlist}) ; display_vars "${_longest}" ${varlist}
  # strip ANSI codes and append to logFile
#  display_vars "${_longest}" ${varlist} | sed 's/\x1b\[[0-9;]*m//g' >> "$logFile"
  display_vars "${_longest}" ${varlist} | _translate_escapes | _strip_ansi
  unset -v _longest; }
# @usage log_show_config
# @vars varlist, logFile, PN, BUILD
# @deps d_log_message log_separator get_longest display_vars _translate_escapes _strip_an
si
# @rule displays colorized config to stdout and sanitized text to $logFile
# @note uses the translation strategy (_strip_ansi) for log file cleanliness

#-----[ Debugging ]-------------------------------------------------------------------
----------

#~~~~~[ basic debiggomg ]~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~
debug_do()          # (POSIX) IAW verbosity, execute one command
{ # @usage debug_do <command string> <level>
  # @note legacy wrapper; redirects to d_do()
  # @warn deprecated (use d_do, which has been upgraded)
  E_message "Warning: deprecated (use d_do, which has been upgraded)"
  d_do "$@"; }

d_do() # (POSIX) IAW verbosity, execute complex (multi-command) jobs
{ # @usage d_do <command string> <level>
  # @args last arg: integer verbosity level; preceding args: the command(s) to execute
  # @deps E_message isint
  # @rule if $verbosity >= $level, executes command(s); otherwise returns 0
  # @rule uses POSIX argument rotation (set --) to separate the level from the command
  # @rule executes via eval if a single string is provided, or directly if multiple args
remain
  # @note handles complex strings like d_do 'ls | grep txt' 2 safely
  # @note detects unset global verbosity and missing arguments via guard clauses

  # guard 1 - check for min. 2 args (word + level)
  [ $# -lt 2 ] && { E_message "minimum 2 args required <command string> <level>"; return
1; }
  # guard 2 - check for unset global verbosity
  [ -z "$verbosity" ] && { E_message "Error: global verbosity not set"; return 2; }
  # extract the last argument and assign to _dd_level POSIX-style
  #   _dd_level will now equal the last argument (equivalent to for _dd_level in "$@"; do
 :; done)
  for _dd_level; do :; done;
```

```
  # guard 3 - ensure _dd_level is an integer
  ! isint "$_dd_level"  && { E_message "Error: Final arg '$_dd_level' must be a decimal v
erbosity level"; return 3; }
  # guard 4 - check global verbosity thresholds
  # @usage d_do <command string> <level>
  # @arg level must be the last argument (is compared to global verbosity)
  # @rule complex multi-command strings like d_do 'ls | grep txt' 2 must be quoted
  # @rule simple (single-command) strings like d_do ls -la 1 can be unquoted
  if [ "$verbosity" -ge "$_dd_level" ]; then
    # argument rotation: assign everything except the last arg back to $@
    _dd_arg_count=$(( $# - 1 ))   # this is the key that will leave the last arg at $1 wh
en the while completes
    _i=0
    while [ "$_i" -lt "$_dd_arg_count" ]; do
       _dd_arg="$1"; shift
       set -- "$@" "$_dd_arg"
       _i=$((_i + 1))
    done
    shift # Remove the arg alread assigned to $_dd_level, which is now at $1
    # safe execution - even for 'd_do "" 1', $#=1 post shift; eval "" does nothing, retur
ns 0 (safe)
    if [ $# -eq 1 ]; then
      eval "$1"  # if user passed a single string with ';' or '&&', eval it.
    else
      "$@"       # Otherwise, execute the arguments directly.
    fi  # zero or one command arg?
  else
    :  # if global verbosity level does not exceed the conditional _dd_level, just return
 0
  fi  # verbosity?
  unset -v _dd_level _dd_arg_count _i _dd_arg
  return 0
}
# In the very specific scenario where a user calls d_do 1 (only one argument),
#   Guard 1 catches it. If they call d_do "" 1 (empty string command),
#   Guard 4 processes it, shift happens, $# becomes 1, and eval "" runs safely

d_message()   # (POSIX) IAW verbosity, display text debugging message $@
{ d_do message "$@" >&2 ; }
# @usage d_message "message string" <level>

d_message_n()   # (POSIX) IAW verbosity, display text debugging message $@ (no CR)
{ d_do message_n "$@" >&2 ; }
# @usage d_message_n "message string" <level>

d_echo()   # (POSIX) IAW verbosity, display text debugging message $@ (like echo)
{ d_do echo "$@" >&2 ; }
# @usage d_echo "message string" <level>

d_echo_n() # (POSIX) IAW verbosity, display text debugging message $@ (like echo -n)
{ d_do echo -n "$@" >&2 ; }
# @usage d_echo_n "message string" <level>

de_echo()   # (POSIX) IAW verbosity, display text debugging message $@ (like echo -e)
{ d_do echo -e "$@" >&2 ; }
# @usage de_echo "message string" <level>

de_echo_n()   # (POSIX) IAW verbosity, display text debugging message $@ (like echo -en)
{ d_do echo -e -n "$@" >&2 ; }
# @usage d_echo_n "message string" <level>

dE_message()   # (POSIX) IAW verbosity, display text debugging error message $@
{ d_do E_message "$@" >&2 ; }
# @usage dE_message "message string" <level>

dE_message_n()   # (POSIX) IAW verbosity, display text debugging error message $@ (w/o CR
)
{ d_do E_message_n "$@" >&2 ; }
# @usage dE_message_n "message string" <level>
```

```
d_right_status()    # output rigit_status IAW verbosity
{ d_do right_status "$@" >&2 ; }
# @usage d_right_status <status_code> <level>

#~~~~~[ debuggimg w logging ]~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~

d_log_separator() # (POSIX) conditionally output separator() to stdout and similar to log
File IAW verbosity
{ d_do log_separator "$@" >&2 ; }
# @usage d_log_separator "preface" "title" <level>

d_log_message()     # (POSIX) display and log message IAW verbosity
{ d_do log_message "$@" >&2 ; }
# @usage d_log_message "message string" <level>

d_log_message_n()     # (POSIX) display and log message IAW verbosity (w/o CR)
{ d_do log_message_n "$@" >&2 ; }
# @usage d_log_message_n "message string" <level>

d_log_echo()        # (POSIX) display and log simple message IAW verbosity (like echo)
{ d_do log_echo "$@" >&2 ; }
# @usage d_log_echo "message string" <level>

d_log_echo_e()      # (POSIX) display and log simple message IAW verbosity (like echo -e)
{ d_do log_echo_e "$@" >&2 ; }
# @usage d_log_echo_e "message string" <level>

d_log_echo_n()      # (POSIX) display and log simple message IAW verbosity (like echo -n)
{ d_do log_echo_n "$@" >&2 ; }
# @usage d_log_echo_n "message string" <level>

d_log_echo_e_n()    # (POSIX) display and log simple message IAW verbosity (like echo -en
)
{ d_do log_echo_e_n "$@" >&2 ; }
# @usage d_log_echo_e_n "message string" <level>

d_log_E_message()   # (POSIX) display and log error message IAW verbosity
{ d_do log_E_message "$@" >&2 ; }
# @usage d_log_E_message "message string" <level>

d_log_E_message_n() # (POSIX) display and log error message IAW verbosity (w/o CR)
{ d_do log_E_message_n "$@" >&2 ; }
# @usage d_log_E_message_n "message string" <level>

d_log_handle_result() # output ok_/_err_msg and right_stats to stdout and logFile
{ d_do log_handle_result "$@" >&2 ; }
# @usage d_log_handle_result "message string" <level>

d_handle_result() # output ok_/_err_msg and right_stats to stdout and logFile
{ d_do handle_result "$@" >&2 ; }
# @usage d_handle_result "message string" <level>

d_log_right_status()# output log_right_status IAW verbosity
{ d_do log_right_status "$@" >&2 ; }
# @usage d_log_right_status "message string" <level>

#=====[ Section 7: System, Net & LUKS ]==================================================
======================

milli_sleep() # (almost POSIX)(busybox safe) "sleep" for sub-second delays
{ # @usage milli_sleep <ms>
  # @args $1: delay in milliseconds (default: 100)
  # @deps awk
  # @note uses awk system() to provide sub-second sleep for shells lacking it
    _ms_in="${1:-100}"
    # pass the variable into awk via -v to avoid shell quoting hell
    # (with -v, the shell variable is passed as data, not as part of the code to be inter
preted)
    # awk then handles the division and calls the system sleep.
```

```
    awk -v ms="$_ms_in" 'BEGIN { system("sleep " ms / 1000) }'
}

fe() # find ebuild files matching filter "*$1*"
{ [ $# -lt 1 ] && { E_message "please specify ebuild patter for search filter"; return 1
; }
  _fe_repo_dir="/var/db/repos/"; _fe_filter="*$1*.ebuild"
  printf "%s\n" "${BYon}looking for ebuild(s) matching filter${Boff}: \"${Mon}${_fe_filte
r}${Boff}\""
  _fe_output=$(find "${_fe_repo_dir}" -iname "${_fe_filter}" -type f -print 2>/dev/null)
  if [ -z "${_fe_output}" ] ; then
    E_message "no files matching ${_fe_filter} found in ${_fe_repo_dir}"
    unset -v _fe_repo_dir _fe_filter _fe_output ; return 1
  else
    printf "%s\n" "${BWon}${_fe_output}${Boff}"
  fi
  unset -v _fe_repo_dir _fe_filter _fe_output
  return 0
}
# @usage fe <pattern>
# @args $1: search pattern for ebuild filenames
# @deps find
# @ret prints matching ebuild paths to stdout; returns 1 if none found
# @note searches recursively within /var/db/repos/

pur() # (POSIX) ping ip address $1 until it is ready (0% loss of 3 packets)
{ _pur_target="$1"; SCP; _pur_count=1; _hits=0; _hit_color="${Boff}"; _pur_max_retries=10
00 ;
  while [ "$_hits" -lt 3 ] && [ "$_pur_count" -lt "$_pur_max_retries" ]; do
    if ping -c1 -W1 "$_pur_target" >/dev/null 2>&1; then _hits=$((_hits + 1)); else _hits
=0; fi;
    case "$_hits" in [01]) _hit_color="${BRon}";; 2) _hit_color="${BYon}";; 3) _hit_color
="${BGon}";; esac;
    RCP; printf '%b' "not ready -- try: [${_pur_count}], hits: [${_hit_color}${_hits}${Bo
ff}] ";
    _pur_count=$((_pur_count + 1)); [ "$_hits" -lt 3 ] && sleep 1; done;
  RCP; EL ; # erase from cursor to end of line
  if [ "$_hits" -lt 3 ] ; then
    RCP; E_message "timed out"; unset -v _pur_target _pur_count _hits _hit_color _pur_max
_retries; return 1 ;
  else
    RCP; message "${BGon}OK${Boff} - ready now" ; unset -v _pur_target _pur_count _hits _
hit_color _pur_max_retries; return 0 ;
  fi
}
# @usage pur <ip_address>
# @args $1: target IP or hostname
# @deps ping SCP RCP EL
# @ret 0 when 3 consecutive pings succeed; 1 on timeout (1000 retries)
# @note uses SCP/RCP for in-place terminal status updates

swr() # (POSIX) use pur() to check first; then ssh when netcat indicates port is ready
{ [ $# -lt 1 ] && return 1 ;   # guard; reject null target
  _swr_target="$1"; _swr_max_retries=10; _swr_count=0; _ssh_port="${2:-22}";  # default t
o port 22 but allow another to be specified as $2
  _yellow_thresh=$(($_swr_max_retries * 33/100)); _red_thresh=$(($_swr_max_retries * 75/1
00));
  pur "$_swr_target";
  if [ ! $? -eq 0 ] ; then
    E_message "pur() failed or timed out";
    unset -v _swr_target _ssh_port _swr_max_retries _swr_count _swr_color _yellow_thresh
_red_thresh
    return 1;
  fi;
  message "confirming $1 has started sshd on port $_ssh_port ..." ; SCP;
  # use netcat to verify port is open on target (one hit is sufficient)
  while [ "$_swr_count" -lt "$_swr_max_retries" ] && ! nc -z -w 1 "$_swr_target" "$_ssh_p
ort" 2>/dev/null; do
    RCP; EL;
    # set colors with POSIX logic
```

```
    if [ "$_swr_count" -le "$_yellow_thresh" ]; then
      _swr_color="${BGon}"
    elif [ "$_swr_count" -le "$_red_thresh" ]; then
      _swr_color="${BYon}"
    else
      _swr_color="${BRon}"
    fi
      printf "checking port %s -- try: [%b] " "$_ssh_port" "${_swr_color}${_swr_count}${B
off}";
    _swr_count=$((_swr_count + 1)); sleep 1; done;
  # test nc one last time to ensure the loop exited for success, not timeout
  if nc -z -w 1 "$_swr_target" "$_ssh_port" 2>/dev/null; then
    RCP; EL; message "${BGon}OK${Boff} - $_swr_target sshd is ready"
    ssh -p "$_ssh_port" "$_swr_target"
    _result=0
  else
    RCP; EL; E_message "sshd port $_ssh_port on $_swr_target not responding after $_swr_m
ax_retries tries"
    _result=1
  fi
  unset -v _swr_target _ssh_port _swr_max_retries _swr_count _swr_color _yellow_thresh _r
ed_thresh
  return $_result
}
# @usage swr <target> [port]
# @args $1: target IP/host; $2: SSH port (default 22)
# @deps pur nc ssh SCP RCP EL
# @ret 0 if SSH connection succeeds; 1 if pur() fails or port never opens
# @rule calls pur() first to ensure basic network reachability
# @note color-coded retry counter (green -> yellow -> red) based on retry thresholds


get_luks_keyfile()          # use arg1 (_glk_crypt_keydev) to identify keyfile
{ # Note: crypt_keydev_mountpoint must be globally defined by the calling script
  [ $# -lt 1 ] && { E_message "please specify crypt_keydev as arg \$1"; return 1 ; }
  _glk_crypt_keydev="$1"
  # ensure trailing slashes are removed for a clean grep match
  _glk_mnt_fixed="${crypt_keydev_mountpoint%/}"
  # determine keydev type
  # if it starts with "/" it is a device name
  # if it contains "-" it may be a UUID
  # if it does not, it may be a LABEL
  message_n "examining _glk_crypt_keydev [ ${_glk_crypt_keydev} ] ..."
  # determine keydev type (POSIX case is faster than grep)
  case "${_glk_crypt_keydev}" in
    /*)   _glk_crypt_keydev_type="devicename" ;;
    *-*)  _glk_crypt_keydev_type="UUID" ;;
    *)    _glk_crypt_keydev_type="LABEL" ;;
  esac
  printf '%b' " (${Mon}${_glk_crypt_keydev_type}${Boff})"; right_status $TRUE
  # if _glk_crypt_keydev is not mounted, try to mount it
  # safe grep: match space-padded " /path " exactly to avoid partial directory false-matc
hes
  # check /proc/mounts directly as the source of truth
  if grep -q " ${_glk_mnt_fixed} " /proc/mounts; then
    printf '%b' " (${BGon}mounted${Boff})"
    right_status $TRUE
  else
    message_n "trying to mount (${_glk_crypt_keydev_type}) [${_glk_crypt_keydev}] ..."
    case $_glk_crypt_keydev_type in
       "devicename" ) mount "${_glk_crypt_keydev}" "${_glk_mnt_fixed}" ;;
       "UUID"       ) mount UUID="${_glk_crypt_keydev}" "${_glk_mnt_fixed}" ;;
       "LABEL"      ) mount LABEL="${_glk_crypt_keydev}" "${_glk_mnt_fixed}" ;;
       * ) printf '%b' " (${BRon}failed${Boff}) ..."; right_status 1 ; die "invalid _glk_
crypt_keydev" ;;
    esac
    _glk_result=$?
    case $_glk_result in
        0 ) printf '%b' " (${BGon}mounted${Boff})"; right_status $TRUE;;
        * ) right_status 1; die "could not mount _glk_crypt_keydev ($_glk_crypt_keydev_ty
```

```
pe)";;
    esac
  fi
  # set keyfile pathname
  export keyfile="${_glk_mnt_fixed}/crypt/dat"
  unset -v _glk_mnt_fixed _glk_crypt_keydev _glk_crypt_keydev_type _glk_result
  return 0
}
# @usage get_luks_keyfile <key_device>
# @args $1: the device containing the key (dev path, UUID, or LABEL)
# @vars crypt_keydev_mountpoint (required global), keyfile (exported)
# @deps mount grep right_status message_n die
# @rule identifies device type (dev/UUID/LABEL) using POSIX case patterns
# @rule verifies mount status via /proc/mounts with space-padding to prevent false matche
s
# @ret 0 on success; calls die on mount failure
# @note Note: crypt_keydev_mountpoint must be globally defined by the calling script
# @note exports the $keyfile variable for use by cryptsetup or other LUKS tools

get_luks_edev_name()       # determine an edev name to use with this device
{ # Note: luks_partition_device must be globally defined
  [ -z "${luks_partition_device}" ] && die "luks_partition_device not defined"
  # ( e.g. sda2->eda2 nvme0n1p2->ev012 mmcblk0p2->em02 )
  _glen_bn="${luks_partition_device##*/}" # equivalent to $(basename ${luks_partition_dev
ice})
  d_message_n "determining edev name ..." 3
  case ${_glen_bn} in
      "sd"* ) # scsi device (sda2 -> eda2)
        # strip s w/ parameter expansion, and replace w/ e
        _glen_myedev="e${_glen_bn#s}"
        ;;
      "nv"* ) # nvme device (nvme0n1p2 -> ev012)
        # strip 'nvme'w/ parameter expansion, then delete the n and the p w/ printf | tr
        _glen_tmp="${_glen_bn#nvme}"
        _glen_myedev="ev$(printf '%s' "$_glen_tmp" | tr -d 'np')"
        ;;
      "mm"* ) # mmc device (mmcblk0p2 -> em02)
         # strip 'mmcblk'w/ parameter expansion, then delete the p w/ printf | tr
        _glen_tmp="${_glen_bn#mmcblk}"
        _glen_myedev="em$(printf '%s' "$_glen_tmp" | tr -d 'p')"
        ;;
      *    ) # tbd
        d_echo "" 3 ; die "bad luks device name [${_glen_bn}]" 3
        ;;
    esac
    _glen_result=$?
    d_echo_n " (${_glen_myedev})" 3 ; d_right_status $TRUE 3
    export edev=${_glen_myedev}
    unset -v _glen_bn _glen_myedev _glen_tmp _glen_result
    return 0
}
# @usage get_luks_edev_name
# @vars luks_partition_device (required global), edev (exported)
# @deps d_message_n d_echo d_echo_n d_right_status tr
# @rule maps physical devices to 'e' prefixed names: sd* -> ed*, nvme* -> ev*, mmcblk* ->
 em*
# @rule uses parameter expansion and tr to sanitize device strings for LUKS mapper
# @ret 0 on success; calls die on unrecognized device patterns
# @note Note: luks_partition_device must be globally defined
# @note example: nvme0n1p2 -> ev012; sda2 -> eda2; mmcblk0p2 -> em02

#=====[ Section 8: Summary & Demonstration Suite ]=====================================
======================


summarize_header() # (POSIX) list functions and notes in installed headers
{ # Note: script_header_installed_dir may be defined in/prior to function call
  #   to enable testing development versions
  # @usage summarize_header [filter_pattern]
  # @args $1: optional regex/string to filter function names (e.g., "log_")
```

```
  # @deps sed grep termwidth
  # @rule processes the script source line-by-line to extract function names and @tags
  # @rule filters output based on $1 to show specific function families
  # @note this is the engine for the 'toc' command and header self-documentation
  # @note future: will be extracted to a stand-alone package for broader script analysis

  # define some regex search patterns --
  # _sh_FUNCTION_REGEX: matches any legal POSIX filename
  # ( must start w alpha char or "_", and may include only alpha, numeric, or "_" prior t
o () )
  _sh_FUNCTION_REGEX="^[_[:alpha:]][_[:alnum:]]*\(\)"
  # _sh_METADATA_REGEX: matches summarize_headers() convention for supplemental comments
for users
  # match our new metadata tag format "# @tagname decriptivetext"
  # so look for zero+ whitespace, followed by one+ #,
  # followed by one+ whitespce, followed by @, followed by one+ alphanum
  _sh_METADATA_REGEX="^${W0}#+${W1}@${A1}"
  # default to all script_header_joetoo modules installed in /usr/sbin/
  if [ -d "$script_header_installed_dir" ] ; then
    _sh_dir="${script_header_installed_dir%/}"
  else
    _sh_dir="/usr/sbin"
  fi
  [ ! -d "$_sh_dir" ] && { E_message "invalid _sh_dir [$_sh_dir]"; return 1 ; }
  if [ $# -eq 0 ] ; then
    _sh_header_glob="${_sh_dir}/script_header_joetoo*"
  else
    _sh_header_glob="${_sh_dir}/script_header_joetoo_${1}*"
  fi
  message "looking for headers in $_sh_header_glob"
  # expand the glob into positional parameters
  # to protect against the remote possibility of filename with whitespace
  set -- ${_sh_header_glob}
  # check if the glob actually found files
  if [ ! -f "$1" ]; then E_message "No header sources found in ${_sh_dir}"; return 1; fi
  # begin output with separator
  separator "summarize_header-${BUILD}" "(content summary)"
  # now that message_n employs echo_n_long, use that to word-wrap this
  _sh_msg="This script header defines ${BYon}common variables${Boff}, ${BYon}ANSI sequenc
es${Boff}, "
  _sh_msg="${_sh_msg}and the following ${BGon}functions/utilities${Boff} as described bel
ow:\n"
  message_n ${_sh_msg}; printf "\n"   # message unquoted to enable word-wrap
  # identify and process all matching installed header files
  for _sh_filespec in "$@"; do
    if [ ! -f "$_sh_filespec" ] ;then
      E_message "$_sh_filespec not found; continuing"
      continue  # if grep id'd a bad filename, skip to next in for loop
    fi
    # generate a separator for each module
    separator "${_sh_PN}-${BUILD}" "($(basename ${_sh_filespec}))"

    # make two passes through the selected header file, use regex-parsing to select all l
ines
    #   that are either a function definition or a metadata tag/paylod
    # in the first pass, calculate the max length of function names and metadata lables u
sed
    #   so as to be able (in the second pass) to determine starting column and indent int
ended
    #     for both function names and metadata
    # in the second pass through the selected header file, use that info and the length o
f
    #   the current function name/metadata label to calculate the dot-padding required to
    #   provide the desired visual alignment, and generate the appropriately colorized ou
tput

    #-----[ FIRST PASS ]------------------------------------------------------------------
    # measure longest function name in this header file (to determine printf column width
)
    # (extract only the function name to a list, and then calculate the longest)
```

```
    _sh_fn_list=$(grep -E "$_sh_FUNCTION_REGEX" "$_sh_filespec" | cut -d'(' -f1)

    # FUNCTIONs
    # with the list, calculate the length of the longest function name
    #   add 2 for "()" to get true max displayed length
    _sh_fn_max_len=$(( $(get_longest $_sh_fn_list) + 2 ))
    d_message "_sh_fn_max_len: $_sh_fn_max_len" 4
    # assign the internal separator to appear between function name and padding
    _sh_fn_sep=" "
    # calculate length of the column (between the gutters) to be passed to pad()-->echo_n
_long()
    _sh_fn_col_width=$(( _sh_fn_max_len + ${#_sh_fn_sep} ))
    # determine the offset to the function description column that accomodates
    # the entire column and the gutters - this is where the cursor
    #   will be when echo_n_long is called to smart-wrap the description
    #   Math: _sh_fn_max_len + 2c.Lpad + 2c.dots(min) + 2c.Rpad
    _sh_fn_col=$((_sh_fn_col_width + 2 + 2 + 2 ))
    d_message "_sh_fn_col: $_sh_fn_col" 4
    # assign the number of the column to which to indent any subseqent lines of the descr
iption
    _sh_fn_indent="${_sh_fn_col}"   # for now, wrap/indent to align directly under start
point

    # METADATA
    # calculate the length of the longest metadata lable to determine the prinf column wi
dth
    #   for metadata (these are known a priori, but change)
    _sh_label_list="usage arguments variables dependencies requirements"
    _sh_label_list="${_sh_label_list} rules notes warning unknown"
    _sh_label_max_len=$(get_longest $_sh_label_list)
    d_message "_sh_label_list: $_sh_label_list" 4
    d_message "_sh_label_max_len: $_sh_label_max_len" 4
    # assign the internal separator to appear between metadata label and padding
    _sh_label_sep=": "
    # calculate the length of the column (between the gutters) to be passed to pad()
    _sh_label_col_width=$(( _sh_label_max_len + ${#_sh_label_sep} ))
    # determine the offset to the function metadata payload column that accomodates
    # the entire column and the gutters - this is where the cursor
    #   will be when echo_n_long is called to smart-wrap the metadata payload
    #   Math: _sh_label_max_len + 4c.Lpad + 2c.dots(min) + 2c.Rpad
    _sh_label_col=$((_sh_label_col_width + 4 + 2 + 2 ))
    d_message "_sh_label_col: $_sh_label_col" 4
    # assign the number of the column to which to indent any subsequent lines of metadata
 payload
    _sh_label_indent="${_sh_label_col}"   # for now, wrap/indent to align directly under
start point

    # NOTE: I am undecided about whether I want to have both
    #   the descrition column and payload columns line up
    #   (that would require one line of math right here, to calculate the difference betw
een
    #    _sh_fn_col and _sh_label_col, to use as an additional offset in _sh_label_col_wi
dth,
    #    which is the width of the metadata label column, thus padding it further to get
    #    the right-side columns all lined up. ok as is, for now

    #-----[ SECOND PASS ]------------------------------------------------------------
    # regex-parse for both '^function()  # comment' and/or '^# *()* comment'
    # ( pick up both function definitions and tagged metadata )
    # ( see regex definitions above )( pipe grep output to a while read loop )
    grep -E "^(${_sh_FUNCTION_REGEX}|${_sh_METADATA_REGEX})" "$_sh_filespec" | while read
 -r _sh_line; do
        # handle POSIX function name
#        d_message "_sh_line: $_sh_line" 4
        # example: _sh_line: SCP()    # (ANSI) save the current cursor position || _sh_fn
_name: [SCP()   ]
        if printf '%s\n' "$_sh_line" | grep -Eq "$_sh_FUNCTION_REGEX"; then
            # check if there is still whitespace - remove if so
            # this printf generates a string w/o CR,
            # piped to tr to delete (-d) the complement (-c) of whitespace
```

```
            # piped to wc to count the characters left
            # i.e. grab filename [text left of first (]
            _sh_tmp=$(printf '%s' "$_sh_line" | cut -d'(' -f1)
            # trim whitespace if needed
            if [ "$(printf '%s' "$_sh_tmp" | tr -dc '[:space:]' | wc -c)" -gt 0 ] ; then
                _sh_tmp="${_sh_tmp#"${_sh_tmp%%[! ]*}"}"              # ltrim (just in ca
se)
                _sh_fn_name="${_sh_tmp%"${_sh_tmp##*[![:space:]]}"}"  # r-trim (just in c
ase)
            else
                _sh_fn_name="$_sh_tmp"   # <=== 99% of the time we probably come right he
re
            fi
            # put the parentheses back on and handle null case
            if [ -z "$_sh_fn_name" ] ; then
              _sh_fn_name="<null>"
            else
              _sh_fn_name="${_sh_fn_name}()"
            fi
#            d_message "_sh_fn_name: [${_sh_fn_name}]" 4
            _sh_comment="${_sh_line#*#}" # all text right of #
            # handle "null comment" cases where no # exists on the line
            [ "$_sh_comment" = "$_sh_line" ] && _sh_comment=""
            # OUTPUT 2c margin gutter pad, colorized function name, and 1c space
            printf '%b' "  ${_func_color}${_sh_fn_name} "
#            printf '%b' "  ${BYon}${_sh_fn_name} "             <--- change from yellow t
o _func_color
#            d_message "_sh_fn_max_len: $_sh_fn_max_len" 4
#            d_message "#_sh_fn_name: ${#_sh_fn_name}" 4
            # OUTPUT dot-padding and gutter padding (2c)
            # use pad() to generate the required padding
            #   usage: pad "cur" "sep" "min" "color" "marker" "col"
            pad "$_sh_fn_name" ' ' "2" "${_func_color}" '.' "${_sh_fn_col_width}"
#            pad "$_sh_fn_name" ' ' "2" "${Yon}" '.' "${_sh_fn_col_width}"        <--- ch
ange from yellow to _func_color
            printf "  "  # inter-column pad spacing
            echo_n_long "$_sh_fn_col" "$_sh_fn_indent" "$_sh_comment"               # sma
rt-word wrap comment
            printf "\n"   # newline for hanging echo_n_long
        # handle metadata tags
        elif printf '%s\n' "$_sh_line" | grep -Eq "$_sh_METADATA_REGEX"; then
            # extract and parse tag (text between '@' and first space)
            _sh_tag=$(printf '%s\n' "$_sh_line" | cut -d'@' -f2 | cut -d' ' -f1)
#            _sh_label_payload=$(printf '%s\n' "$_sh_line" | cut -d'@' -f2 | cut -d' ' -f
2-)
             # sed matches (and substitutes a null string for) i.e. removes -
             #   line beginning with zero+ whitespace, followed by #, followed by
             #   one+ whitespace, followed by @ (the tag symbol), followed by
             #   the tag lable just extracted in the command above, followed by
             #   one+ whitespace
            _sh_label_payload=$(printf '%s\n' "$_sh_line" | sed "s/^${W0}#${W1}@${_sh_tag
}${W1}//")
            case "$_sh_tag" in
                usage) _sh_label="usage" ;;
                args ) _sh_label="arguments" ;;
                vars ) _sh_label="variables" ;;
                ret  ) _sh_label="returns" ;;
                deps ) _sh_label="dependencies" ;;
                req  ) _sh_label="requirements" ;;
                rule ) _sh_label="rules" ;;
                note ) _sh_label="notes" ;;
                warn ) _sh_label="warning" ;;
                ex   ) _sh_label="example" ;;
                cont ) _sh_label="continuation" ;;
                ''   ) _sh_label="<null>" ;;
                *    ) _sh_label="unknown" ;;
            esac
            # note: longest of these is 13 char
#            d_message "_sh_label: $_sh_label" 4
#            d_message "_sh_label_col: $_sh_label_col" 4
```

```
#            d_message "_sh_label_indent: $_sh_label_indent" 4
            # OUTPUT 4c margin gutter pad, colorized function name, and 1c space
            printf "    ${_data_color}%-${#_sh_label}s:${Boff} " "$_sh_label"
#            printf "    ${BMon}%-${#_sh_label}s:${Boff} " "$_sh_label"         <--- chan
ge from $BMon to _data_color
            # OUTPUT dot-padding and gutter padding (2c)
            # use pad() to generate the required padding
            #   usage: pad "cur" "sep" "min" "color" "marker" "col"
            pad "$_sh_label" ': ' "2" "${_data_color}" '.' "${_sh_label_col_width}"
#            pad "$_sh_label" ': ' "2" "${Mon}" '.' "${_sh_label_col_width}"    <--- chan
ge from Mon to _data_color
            printf '%s' "  " # inter- column gutter

#            d_message "_sh_line: [$_sh_line]" 4
#            d_message "_sh_tag: [$_sh_tag]" 4
#            d_message "_sh_label_payload: [$_sh_label_payload]" 4

            echo_n_long "${_sh_label_col}" "${_sh_label_indent}" "$_sh_label_payload"
            printf "\n"   # newline for hanging echo_n_long
        else  # safety catch for unexpected matches
        printf "  ${BRon}UNKNOWN:  %s${Boff}\n" "$_sh_line"
        fi  # function|comment|other
    done  # while
  done  # for
  echo
  message "To run functional tests, use: ${BGon}demonstrate_header${Boff}"
  separator "$(hostname)" "(complete)"
  unset -v _sh_PN _sh_msg _sh_filespec _sh_line _sh_fn_name _sh_comment _sh_dir _sh_heade
r_glob
  unset -v _sh_fn_list _sh_fn_max_len _sh_FUNCTION_REGEX _sh_fn_col _sh_METADATA_REGEX
  unset -v _sh_label_list _sh_label_max_len _sh_tag _sh_label_col _sh_label_indent _sh_la
bel _sh_label_payload
  unset -v _sh_tmp _sh_fn_sep _sh_fn_indent _sh_fn_col_width _sh_label_sep _sh_label_col_
width
  return 0
}

#-----[ Demonstration Sub-function Stubs ]------------------------------------------------
----------------------

_dh_show_capabilities()   # demonstrate classes of capability in this header file
{ # @args none
  # @rules called by demonstrate_header

# preview the command syntax
  # should print two lines like this but theme-colored --
  # message "Description of header capabilities..."
  # printf "   "; message_n ""; echo_n_long 7 7 "${BCon}Section: content..."
  # this requires a complex sequence of commands, so build it by line and parts
  # line 1 - message "Description of header capabilities..."
  _dh_cmd="${_func_color}message${_Boff} ${_quote_color}\"Description of header capabilit
ies...\"${_Boff}\n"
  # line 2 part 1 - printf "   ";    # note ${_Boff} ${_quote_color} in case not bold
  _dh_cmd="${_dh_cmd} ${_cmd_color}printf${_Boff} ${_quote_color}\"    \"${_Boff};"
  # line 2 part 2 - message_n ""     # message_n displayed as a function
  _dh_cmd="${_dh_cmd} ${_func_color}message_n${_Boff} ${_quote_color}\"\"${_Boff};"
  # line 2 part 3 - echo_n_long 7 7  # ${_Boff} ${_data_color} in case not bold
  _dh_cmd="${_dh_cmd}  ${_func_color}echo_n_long${_Boff} ${_data_color}7 7${_Boff}"
  # line 2 part 4 - "${BCon}Section: content..."
  _dh_cmd="${_dh_cmd} ${_quote_color}\"\${BCon}Section: content...\"${_Boff}"

  printf "\n"; message "${BYon}About to run:${Boff}\n ${_dh_cmd}"; printf "\n"
  # execute the actual demonstration
  _dh_msg="This script header is a robust ${BYon}POSIX-compliant utility suite${Boff}. It
s core capabilities include:"
  message_n ""; echo_n_long 3 3 "${_dh_msg}"
  printf "\n"

  # Variable Definitions
  _dh_msg="${BCon}Variable Definitions:${Boff} Constants for error messages, pseudo-boole
```

```
ans "
  _dh_msg="${_dh_msg}(${BYon}TRUE=0/FALSE=\"\"${Boff}), and terminal state defaults."
  printf "    "; message_n ""; echo_n_long 7 7 "${_dh_msg}"; printf "\n"

  # Validation Functions
  _dh_msg="${BCon}Validation Functions:${Boff} Logic to verify specific numeric types lik
e integers, "
  _dh_msg="${_dh_msg}hex values, and floats (${BYon}isint, ishexint, isfloat_posix${Boff}
), consolidated in ${BYon}isnumeric${Boff}."
  printf "    "; message_n ""; echo_n_long 7 7 "${_dh_msg}"; printf "\n"

  # System and Shell Checks
  _dh_msg="${BCon}System and Shell Checks:${Boff} Tools to verify root status (${BYon}che
ckroot${Boff}), "
  _dh_msg="${_dh_msg}specific mount points (e.g., ${BYon}checkboot${Boff}), and shell int
eractivity."
  printf "    "; message_n ""; echo_n_long 7 7 "${_dh_msg}"; printf "\n"

  # ANSI Escape Sequences
  _dh_msg="${BCon}ANSI Escape Sequences:${Boff} A comprehensive set of variables for text
 styling "
  _dh_msg="${_dh_msg}(colors, bold, underline, reverse) and cursor manipulation (${BYon}H
CU/SCU, SCP/RCP, CUP${Boff})."
  printf "    "; message_n ""; echo_n_long 7 7 "${_dh_msg}"; printf "\n"

  # Messaging Tier
  _dh_msg="${BCon}Messaging Tier:${Boff} A categorized system including standard (${BYon}
message${Boff}), "
  _dh_msg="${_dh_msg}warning (${BYon}W_message${Boff}), and error (${BYon}E_message${Boff
}) outputs with visual color-coded distinction."
  printf "    "; message_n ""; echo_n_long 7 7 "${_dh_msg}"; printf "\n"

  # Box Drawing
  _dh_msg="${BCon}Box Drawing:${Boff} Mappings for DEC Special Graphics characters (e.g.,
 "
  _dh_msg="${_dh_msg}${BYon}H_wall='q', V_wall='x', corners${Boff}) and state control (${
BYon}ESCon, SO, SI${Boff})."
  printf "    "; message_n ""; echo_n_long 7 7 "${_dh_msg}"; printf "\n"

  # Debugging and Logging
  _dh_msg="${BCon}Debugging and Logging:${Boff} Conditional execution/messaging based on
verbosity "
  _dh_msg="${_dh_msg}levels and timestamped file logging functions (e.g., ${BYon}d_log_me
ssage${Boff})."
  printf "    "; message_n ""; echo_n_long 7 7 "${_dh_msg}"; printf "\n"
  printf "\n"
  unset -v _dh_cmd _dh_msg
  return 0
}

_dh_demo_verbose() # (POSIX) demonstrate the use of status_color, TrueFalse, to display s
ystem verbosity
{ # @usage _dh_demo_verbose
  # @args none
  # @rules called by demonstrate_header

  separator "demonstrate_header-${BUILD}" "(verbosity)"

  # preview the command syntax
  # Construct the "About to run" preview using the theme engine. it should like like this
, but theme-colored:
  #   message "${BYon}VERBOSE is currently $(status_color $VERBOSE)$(TrueFalse $VERBOSE)$
{Boff}
  # this involves complex nested expansions, so do it in three pieces we can maintain --
  #   part 1 - message "${BYon}VERBOSE is currently "
  _dh_cmd="${_func_color}message${_Boff} ${_quote_color}\"${_var_color}\${BYon}${_quote_c
olor}VERBOSE is currently "
  #   part 2 - $(status_color $VERBOSE)
  _dh_cmd="${_dh_cmd}\$(${_Boff}${_func_color}status_color${_Boff} ${_var_color}\$VERBOSE
${_Boff}${_quote_color})${_Boff}"
```

```
  #   part 3 $(TrueFalse $VERBOSE)${Boff}
  _dh_cmd="${_dh_cmd}${_quote_color}\$(${_Boff}${_func_color}TrueFalse${_Boff}"   # note
\$ is for subshell
  _dh_cmd="${_dh_cmd} ${_var_color}\$VERBOSE${_Boff}${_quote_color})${_Boff}"     # note
${_Boff}${_quote_color}in case not bold
  _dh_cmd="${_dh_cmd}${_var_color}\${Boff}${_quote_color}\"${_Boff}"              # note
\$VERBOSE, \${Boff} displayed as a variables
  #   also add administrative comment
  _dh_cmd="${_dh_cmd}  ${_cmt_color}# using two-pass column alignment and pad() utility${
_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff}\n ${_dh_cmd}"; printf "\n"

  # execution logic - store current state to avoid side effects in the main script
  _dh_oldV=$VERBOSE; _dh_oldv=$verbosity
  # Set state for demonstration
  VERBOSE=$TRUE; verbosity=3
  message "  ${BYon}VERBOSE is currently $(status_color $VERBOSE)$(TrueFalse $VERBOSE)${B
off}"
  _dh_result=$?
  # Restore state
  VERBOSE=$_dh_oldV; verbosity=$_dh_oldv
  printf "\n"

  unset -v _dh_cmd _dh_oldV _dh_oldv
  return $_dh_result
}

_dh_demo_versions()
{ # @usage _dh_demo_versions
  # @args none
  # @rules called by demonstrate_header

  separator "demonstrate_header-${BUILD}" "(version comparison)"

  _dh_FAILED=$FALSE
  _dh_ret=0

# preview the command syntax
  _dh_cmd="${_ctl_color}for${_Boff} ${_var_color}v${_Boff} ${_ctl_color}in${_Boff} ${_quo
te_color}\"1 1\" \"2 1\" \"1 2\" ...${_Boff}; ${_ctl_color}do${_Boff}\n"
  _dh_cmd="${_dh_cmd}   ${_func_color}vercomp${_Boff} ${_data_color}\$v1 \$v2${_Boff}; ${
_func_color}show_result${_Boff} ${_data_color}\$?${_Boff}\n"
  _dh_cmd="${_dh_cmd} ${_ctl_color}done${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff}\n ${_dh_cmd}"; printf "\n"

  # execute the actual demonstration
  for _dh_vtest in "1 1" "2 1" "1 2" "1.2.0 1.2.1" "1.2.0-r1 1.2.0"; do
    _dh_v1=${_dh_vtest% *}  # equiv =$( printf '%s' "${_dh_vtest}" | cut -d' ' -f1) (but
w/o call to cut)
    _dh_v2=${_dh_vtest#* }  # equiv =$( printf '%s' "${_dh_vtest}" | cut -d' ' -f2) (but
w/o call to cut)

    message_n "${BYon}Compare version ${BWon}${_dh_v1} ${BYon}to ${BWon}${_dh_v2}${Boff}
 "
    vercomp "$_dh_v1" "$_dh_v2"
    _dh_vercomp_result=$?
    printf "%b\n" "$(show_result $_dh_vercomp_result)"
    show_result $_dh_vercomp_result >/dev/null 2>&1
    _dh_show_result_exit=$?
    # if any single comparison fails (returns non-zero), mark the module as failed
    [ "$_dh_show_result_exit" -eq 0 ] || _dh_FAILED=$TRUE
  done
  printf "\n"

  [ "$_dh_FAILED" ] && _dh_ret=1
  unset -v _dh_cmd _dh_vtest _dh_v1 _dh_v2 _dh_vercomp_result _dh_show_result_exit _dh_FA
ILED
  return "$_dh_ret"
}
```

```
_dh_demo_numeric() # (POSIX) demonstrate numeric validation utilities
{ # @usage _dh_demo_numeric
  # @args none
  # @rules called by demonstrate_header

  separator "demonstrate_header-${BUILD}" "(numeric validation)"

  # preview the command syntax
  _dh_cmd="${_func_color}isnumeric${_Boff} ${_quote_color}\"123.45\"${_Boff}${_op_color};
${_Boff} ${_func_color}isint${_Boff} ${_quote_color}\"123\"${_Boff}   "
  _dh_cmd="${_dh_cmd} ${_cmt_color}# using two-pass column alignment and pad() utility${_
Boff}"
  printf "\n"; message "${BYon}About to run:${Boff}\n ${_dh_cmd}"; printf "\n"

  message "${BYon}Demonstrating Numeric Validation Utilities --${Boff}"

  _dh_tests="isint|123|0 isint|abc|1 isfloat_posix|3.14|0 isfloat_posix|123|1 ishexint|0x
AF|0 ishexint|123|1 isnumeric|42.5|0 isnumeric|0xCC|0 isnumeric|xyz|1"
  _dh_sep=" "; _dh_sep_len=${#_dh_sep}; _dh_min=2
  _dh_FAILED=$FALSE
  _dh_ret=0

  # notes on pad() usage: pad "cur" "sep" "min" "color" "marker" "col"
  # (  need to make 2 passes to parse the test data
  #     - one to calculate fixed targets for padding
  #     - second to output each line, properly formatted
  # pad() api --
  # req all 6 args must be supplied; see below how some '' vals can default
  # note where . str cur = the current entry in fixed width column
  # note ....... str sep = internal separator (e.g. ': ')
  # note ....... int min = the minimum number of pad chars ...... (default 2)
  # note ..... str color = ANSI escape sequence (e.g. '\033[32m')
  # note ... char marker = the character w which to construct pad (default '.')
  # note ....... int col = the columns total fixed width
  # note marker may be colorized w ANSI sequence in quoted arg
  # warn failure to properly quote all args may result in unexpected behavior

  # first pass: calculate max lengths
  _dh_max_test_len=0; _dh_max_val_len=0
  for _dh_t in ${_dh_tests}; do
    _dh_test=$(echo "${_dh_t}" | cut -d'|' -f1)
    [ "${#_dh_test}" -gt "$_dh_max_test_len" ] && _dh_max_test_len=${#_dh_test}
    _dh_val=$(echo "${_dh_t}" | cut -d'|' -f2)
    [ "${#_dh_val}" -gt "$_dh_max_val_len" ] && _dh_max_val_len=${#_dh_val}
  done

#  d_message "_dh_max_test_len:  $_dh_max_test_len" 4
#  d_message "_dh_max_val_len: $_dh_max_val_len" 4

  # set fixed widths
  _dh_opening="Testing"; _dh_open_len=${#_dh_opening}
#  d_message "_dh_open_len: $_dh_open_len" 4
#  d_message "_dh_sep_len: $_dh_sep_len" 4
  _dh_test_total_sep=$(( _dh_sep_len * 2 ))
#  d_message "_dh_test_total_sep: $_dh_test_total_sep" 4
  _dh_col_test=$(( $_dh_open_len + $_dh_max_test_len + $_dh_test_total_sep + $_dh_min ))
 #  "Testing isnumeric "
  _dh_col_val=$(( $_dh_max_val_len + $_dh_sep_len + $_dh_min ))  #  "'0xAF' "
  _dh_col_TF=9    #  "[False] " - permanently known emprically ([+F+a+l+s+e+]+.+.+)
#  d_message "_dh_col_test: $_dh_col_test" 4
#  d_message "_dh_col_val: $_dh_col_val" 4
#  d_message "_dh_col_TF: $_dh_col_TF" 4

  # second pass: execution and aligned table print
  for _dh_t in ${_dh_tests}; do
    _dh_test=$(echo "${_dh_t}" | cut -d'|' -f1)
    _dh_val=$(echo "${_dh_t}" | cut -d'|' -f2)
    _dh_exp=$(echo "${_dh_t}" | cut -d'|' -f3)

    ${_dh_test} "${_dh_val}" >/dev/null 2>&1
```

```
    _dh_result=$?

    # determine True/False state for visual coloring
    case "$_dh_result" in
        0 ) _dh_RESULT_BOOL=$TRUE ;;
        * ) _dh_RESULT_BOOL=$FALSE ;;
    esac
    _dh_bool_str="$(TrueFalse $_dh_RESULT_BOOL)"
    _dh_bool_color="$(status_color $_dh_RESULT_BOOL)"

    # determine if actual result matches expected result
    if [ "${_dh_result}" -ne "${_dh_exp}" ]; then
       _dh_FAILED=$TRUE
       _dh_pass_fail="${BRon}FAIL${Boff}"
    else
       _dh_pass_fail="${BGon}PASS${Boff}"
    fi

    # column 1: test function name
    _dh_col1_text="Testing $_dh_test "
    message_n "Testing ${BCon}${_dh_test}${Boff} "
    # force min pad 0 on this column
    pad "${_dh_col1_text} " "$((_dh_sep_len * 3))" "0" "$BCon" "." "$_dh_col_test"

    # column 2: value segment
    _dh_v_label="'${_dh_val}'"
    printf '%s' "'${BMon}${_dh_val}${Boff}' "
    pad "$_dh_v_label" "$_dh_sep_len" "$_dh_min" "$BCon" "." "$_dh_col_val"

    # column 3: status segment (coloring bracketed string correctly)
    printf '%s' " [${_dh_bool_color}${_dh_bool_str}${Boff}] "
    pad " [${_dh_bool_str}]" "$_dh_sep_len" "$_dh_min" "$BCon" "." "$_dh_col_TF"

    # column 4: raw return and final judgement
    printf '%s' " (ret:${_dh_bool_color}${_dh_result}${Boff}|${BWon}${_dh_exp}${Boff}:exp
) -> "
    printf "%s\n" "${_dh_pass_fail}"
  done
  printf "\n"

  [ "$_dh_FAILED" ] && _dh_ret=1
  unset -v _dh_cmd _dh_tests _dh_sep _dh_sep_len _dh_min _dh_max_test_len _dh_max_val_len
  unset -v _dh_opening _dh_open_len _dh_test_total_sep _dh_col_test _dh_col_val _dh_col_T
F
  unset -v _dh_t _dh_test _dh_val _dh_exp _dh_result _dh_RESULT_BOOL _dh_bool_str
  unset -v _dh_bool_color _dh_pass_fail _dh_v_label _dh_col1_text _dh_FAILED

  return "$_dh_ret"
}

_dh_demo_debug_utils() # (POSIX) demonstrate debugging (verbosity-conditional execution a
nd messaging)
{ # @usage _dh_demo_debug_utils
  # @args none
  # @rules demonstrating conditional messaging and status utilities
 _dh_FAILED=$FALSE
  _dh_ret=0

  separator "demonstrate_header-${BUILD}" "(debugging utilities)"

  # temporarily set system verbosity for demonstration
  _dh_oldV=$VERBOSE; _dh_oldv=$verbosity
  VERBOSE=$TRUE; verbosity=3
  message "${BYon}VERBOSE is $(status_color $VERBOSE)$(TrueFalse $VERBOSE)${Boff}"
  message "${BYon}system verbosity is ${Boff}[${_data_color}${verbosity}${_Boff}]"

  # demonstrate d_echo - debugging message (output if verbosity >= level)
  _dh_cmd="${_func_color}d_echo${_Boff} ${_quote_color}\"Raw debug data: \${_var}\"${_Bof
f} ${_data_color}4${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff}  ${_cmt_color}# if verbosity >= 4 (it
```

```
 is not)${Boff}\n ${_dh_cmd}"
  d_echo "  Raw debug data: example_value" 4
  # logic execution - and failure check
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  # demonstrate d_message - debugging message (output if verbosity >= level)
  _dh_cmd="${_func_color}d_message${_Boff} ${_quote_color}\"Debug message at level 3\"${_
Boff} ${_data_color}3${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff}  ${_cmt_color}# if verbosity >= 1 (it
 is)${_Boff}\n ${_dh_cmd}"
  # logic execution - and failure check
  d_message "Debug message at level 3" 3
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  # demonstrate d_message_n; logic test; d_right_status $? (conditional status linkage)
  # case A: looking for a file that does not exist
  _dh_file_bogus="/usr/sbin/script_header_joetoo_bogus"
  # construct line 1 of the commands to be displayed
  _dh_cmd="${_var_color}header${_Boff}=${_quote_color}\"${_dh_file_bogus}\"${_Boff}\n"
  # construct line 2 of the commands to be displayed
  _dh_cmd="${_dh_cmd} ${_func_color}d_message_n${_Boff} "
  _dh_cmd="${_dh_cmd}${_quote_color}\"looking for \${header}\"${_Boff} "
  _dh_cmd="${_dh_cmd}${_data_color}1${_Boff}\n"
  # construct line 3 of the commands to be displayed - starting with the [ test
  _dh_cmd="${_dh_cmd} ${_ctl_color}[${_Boff} ${_op_color}!${_Boff} ${_opt_color}-z${_Boff
} "
  # Add the find subshell
  _dh_cmd="${_dh_cmd}${_quote_color}\"\$(${_cmd_color}find${_Boff} \${header} "
  _dh_cmd="${_dh_cmd}${_data_color}2${_Boff}${_op_color}>/dev/null)${_quote_color}\" "
  # Close bracket and add d_right_status
  _dh_cmd="${_dh_cmd}${_ctl_color}]${_Boff}${_op_color};${_Boff} "
  _dh_cmd="${_dh_cmd}${_func_color}d_right_status${_Boff} ${_var_color}\$?${_Boff} "
  _dh_cmd="${_dh_cmd}${_data_color}1${_Boff}"

  printf "\n"; message "${BYon}About to run:${Boff}  ${_cmt_color}# if verbosity >= 1 (it
 is)${Boff}\n ${_dh_cmd}"
  # logic execution - and failure check
  header="${_dh_file_bogus}"; d_message_n "looking for ${header}" 1
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE
  [ ! -z "$(find ${header} 2>/dev/null)" ]; d_right_status $? 1
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  # case B: Looking for a file that does exist
  _dh_file_real="/usr/sbin/script_header_joetoo"
  # construct line 1 of the commands to be displayed
  _dh_cmd="${_var_color}header${_Boff}=${_quote_color}\"${_dh_file_real}\"${_Boff}\n"
  # construct line 2 of the commands to be displayed
  _dh_cmd="${_dh_cmd} ${_func_color}d_message_n${_Boff} "
  _dh_cmd="${_dh_cmd}${_quote_color}\"looking for \${header}\"${_Boff} "
  _dh_cmd="${_dh_cmd}${_data_color}1${_Boff}\n"
  # construct line 3 of the commands to be displayed - starting with the [ test
  _dh_cmd="${_dh_cmd} ${_ctl_color}[${_Boff} ${_op_color}!${_Boff} ${_opt_color}-z${_Boff
} "
  # Add the find subshell
  _dh_cmd="${_dh_cmd}${_quote_color}\"\$(${_cmd_color}find${_Boff} \${header} "
  _dh_cmd="${_dh_cmd}${_data_color}2${_Boff}${_op_color}>/dev/null)${_quote_color}\" "
  # Close bracket and add d_right_status
  _dh_cmd="${_dh_cmd}${_ctl_color}]${_Boff}${_op_color};${_Boff} "
  _dh_cmd="${_dh_cmd}${_func_color}d_right_status${_Boff} ${_var_color}\$?${_Boff} "
  _dh_cmd="${_dh_cmd}${_data_color}1${_Boff}"

  printf "\n"; message "${BYon}About to run:${Boff}  ${_cmt_color}# if verbosity >= 1  (i
t is)${Boff}\n ${_dh_cmd}"
  # logic execution - and failure check
  header="${_dh_file_real}"; d_message_n "looking for ${header}" 1
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE
  [ ! -z "$(find ${header} 2>/dev/null)" ];   d_right_status $? 1
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  # demonstrate d_do - case A: simple command (execution if verbosity >= level)
```

```
  # build command - d_do "printf '${BGon}Executed${Boff}'" 1 (theme-colored)
  _dh_cmd="${_func_color}d_do${_Boff} ${_quote_color}\"printf '\${BGon}Executed\${Boff}'\
"${_Boff} ${_data_color}1${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff}  ${_cmt_color}# if verbosity >= 1 (it
 is)${_Boff}\n ${_dh_cmd}"
  # logic execution - and failure check
  d_do "printf '${BGon} * Executed${Boff}\n'" 1
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  # demonstrate d_do - case B: complex multi-part command (execution if verbosity >= leve
l)
  # build command d_do 'echo "      line 1.1"; echo "      line 1.2"' 2 (theme colored)
  _dh_cmd="${_func_color}d_do${_Boff} ${_quote_color}'echo \"      line 1.1\"; echo \"
   line 1.2\"'${_Boff} ${_data_color}2${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff}  ${_cmt_color}# if verbosity >= 1 (it
 is)${Boff}\n ${_dh_cmd}"
  # logic execution - and failure check
  d_do 'echo "   line 1.1"; echo "   line 1.2"' 2
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  [ "$_dh_FAILED" = "$TRUE" ] && _dh_ret=1
  # restore previous system verbosity
  VERBOSE=$_dh_oldV; verbosity=$_dh_oldv
  printf "\n"

  unset -v _dh_cmd _dh_FAILED _dh_oldV _dh_oldv _dh_file_bogus _dh_file_real header
  return $_dh_ret
}


_dh_demo_boolean() # (POSIX) demonstrate bool truth validation with a variety of data typ
es
{ # @usage _dh_demo_boolean
  # @args none
  # @rules demonstrating truth-testing and conversion helpers

  _dh_ret=0
  _dh_FAILED=$FALSE

  separator "demonstrate_header-${BUILD}" "(boolean utilities)"

  # data format: label|value|expectation (0=True, 1=False, 2=unset)
  _dh_tests="\$TRUE|$TRUE|0 \$FALSE|$FALSE|1 y|y|0 Y|Y|0"
  _dh_tests="${_dh_tests} n|n|1 N|N|1 t|t|0 T|T|0 f|f|1 F|F|1"
  _dh_tests="${_dh_tests}  up|up|0 down|down|1 0|0|0 1|1|2 z|z|2 ''|''|1"

  # preview the command syntax for each of the commands to be run (4 lines, like this)
  #  for x in "$TRUE $FALSE y ... '' $TRUE $FALSE"; do
  #    message_n "testing: [${x}] is $(status_color ${x})$(TrueFalse ${x})"
  #    if [ "$(TrueFalse ${x})" = "True" ]; then right_status $TRUE
  #    else right_status 1; fi
  #  done
  # line 1 part 1 - for x in
  _dh_cmd="${_ctl_color}  for${_Boff} ${_var_color}x${_Boff} ${_ctl_color}in${_Boff} "
  # line 1 part 2 - "y n 0 1 z ... '' $TRUE $FALSE"; do
  _dh_cmd="${_dh_cmd}${_quote_color}\"\$TRUE \$FALSE y n ... ''\"${_Boff}${_op_color};${_
Boff} ${_ctl_color}do${_Boff}\n"
  # line 2 part 1 -  message_n "testing: [${x}] is
  _dh_cmd="${_dh_cmd}     ${_func_color}message_n${_Boff} ${_quote_color}\"testing: [\${x
}] is \$(${_Boff}"
  # line 2 part 2 - $(status_color ${x})
  _dh_cmd="${_dh_cmd}${_func_color}status_color${_Boff} ${_var_color}\${x}${_Boff}${_quot
e_color})\$(${_Boff}"
  # line 2 part 3 - $(TrueFalse ${x})"
  _dh_cmd="${_dh_cmd}${_func_color}TrueFalse${_Boff} ${_var_color}\${x}${_Boff}${_quote_c
olor})\"${_Boff}\n"
  # line 3 part 1 - if [
  _dh_cmd="${_dh_cmd}     ${_ctl_color}if [${_Boff}"
  # line 3 part 2 - "$(TrueFalse ${x})"
  _dh_cmd="${_dh_cmd}  ${_quote_color}\"\$(${_Boff}${_func_color}TrueFalse${_Boff} ${_var
```

```
_color}\${x}${_Boff}${_quote_color})\"${_Boff} "
  # line 3 part 3 - = "True" ];
  _dh_cmd="${_dh_cmd}${_op_color}=${_Boff} ${_quote_color}\"True\"${_Boff} ${_ctl_color}]
${_Boff}${_op_color};${_Boff} "
  # line 3 part 4 - then right_status $TRUE
  _dh_cmd="${_dh_cmd}${_ctl_color}then${_Boff} ${_func_color}right_status${_Boff} ${_var_
color}\$TRUE${_Boff}\n"
  # line 4 part 1 - else right_status 1;
  _dh_cmd="${_dh_cmd}   ${_ctl_color}else${_Boff} ${_func_color}right_status${_Boff} ${_d
ata_color}1${_Boff}${_op_color};${_Boff} "
  # line 4 part 2 - fi
  _dh_cmd="${_dh_cmd}${_ctl_color}fi${_Boff}\n"
  # line 5 - done
  _dh_cmd="${_dh_cmd}${_ctl_color}  done${_Boff}"

  printf "\n"; message "${BYon}About to run:${Boff}\n ${_dh_cmd}"
  printf "\n"
  message "  ${BYon}Summary of T/F tests with ${BGon}status_color${Boff}() and ${BGon}Tru
eFalse${Boff}() --"

  # logic execution
  for _dh_t in ${_dh_tests}; do
    _dh_label=$(printf '%s' "${_dh_t}" | cut -d'|' -f1)
    _dh_val=$(printf '%s' "${_dh_t}" | cut -d'|' -f2)
    _dh_exp=$(printf '%s' "${_dh_t}" | cut -d'|' -f3)

    # explicit translation of the literal '' placeholder to a true null value
    [ "${_dh_val}" = "''" ] && _dh_val=""
    # get the formatted output
    _dh_out=$(TrueFalse "${_dh_val}")
    # get the exit status from TrueFalse
    _dh_status=$?

    message_n "    ${BWon}testing: [${BCon}${_dh_label}${Boff}${BWon}] is $(status_color
"${_dh_val}")${_dh_out}${Boff}"

    # validate that the exit status matches our expectation for this value
    if [ "$_dh_status" -eq "$_dh_exp" ] ; then
      right_status $TRUE
    else
      right_status 1
      _dh_FAILED=$TRUE
    fi
  done

  printf "\n"
  E_message_n ""
  echo_n_long 3 "" "Note: most shells evaluate null ('') as ${BRon}FALSE${Boff} and non-n
ull as ${BGon}TRUE${Boff}"
  printf "\n"

  [ "$_dh_FAILED" = "$TRUE" ] && _dh_ret=1

  unset -v _dh_cmd _dh_FAILED _dh_t _dh_label _dh_val _dh_exp _dh_out _dh_status _dh_test
s _dh_PN
  return $_dh_ret
}

_dh_demo_terminal_io() # (POSIX) demonstrate terminal io, animation, box drawing
{ # @usage _dh_demo_terminal_io
  # @args none
  # @rules demonstrating cursor state and DEC graphics
  _dh_ret=0
  _dh_FAILED=$FALSE
  _dh_cock="${BRon}-*-${Boff}"    # define a 3-char shuttlecock (stationary)
  _dh_r_cock="${BRon}--*${Boff}"  # define a 3-char shuttlecock (right bound)
  _dh_l_cock="${BRon}*--${Boff}"  # define a 3-char shuttlecock (left bound)
  _dh_wait="0.1"                  # define the loop wait time (seconds)
  _dh_wait="25"                   # define the loop wait time (milliseconds)
  separator "demonstrate_header-${BUILD}" "(terminal i/o & graphics)"
```

```
  #-----[ part 1: shuttle animation ]-----------------------------------
  # ( including hide/show cursor )
  # preview the command syntax for each of the commands to be run (4 lines, like this)
  # construct the command to be displayed - 2 lines
  #    message_n "Shuttle: [                    ]"; CUB 21
  #    # Loop: CUF/CUB to shift '-*-' shuttlecock; HCU/SCU to hide/show cursor
  # line 1 part 1 -    message_n "Shuttle:
  _dh_cmd="${_func_color}message_n${_Boff} ${_quote_color}\"Shuttle:"
  # line 1 part 2  [                    ]";
  _dh_cmd="${_dh_cmd} [                    ]\"${_Boff}${_op_color};${_Boff}"
  # line 1 part3 -  CUB 21
  _dh_cmd="${_dh_cmd} ${_func_color}CUB${_Boff} ${_data_color}21${_Boff}\n"
  # line 2 - # Loop: CUF/CUB to shift '-*-' shuttlecock; HCU/SCU to hide/show cursor
  _dh_cmd="${_dh_cmd} ${_cmt_color}# Loop: CUF/CUB to shift '-=-' shuttlecock; HCU/SCU to
 hide/show cursor${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff} ${_cmt_color}# 20-char window, 2 cycl
es${_Boff}\n ${_dh_cmd}"

  # print the 20 char shuttle field bounded by braces [ ], (22 char total), and return to
 start of field
  # create room in case the cursor is at the bottom of the screen (newline twice then up
1)
  printf "\n\n"; CUU 1

  HCU   # hide the cursor during animation (very distracting otherwise)
  # complete 2 cycles of move to the right and back to the left
  _dh_cycle=0
#  while [ $_dh_cycle -lt 2 ]; do
  while [ "$_dh_cycle" -lt 1 ]; do    # decided to speed things up
    # return to left margin (resets every thing (more important in second+ cycle)
    printf "\r"
    message_n "shuttle zone: [                    ]"
    # return to start of shuttle-zone
    CUB 21
    #draw the shuttlecock
    printf '%s' "${_dh_cock}"; milli_sleep "${_dh_wait}"; CUB 3
    # note: starting cursor location (col the NEXT print goes into) is [ + 4
    _dh_i=0
    while [ "$_dh_i" -lt 17 ]; do     # 21 - 4 = 17 is collision with ]
    # shift shuttle right
      printf '%s' " ${_dh_r_cock}"; milli_sleep "${_dh_wait}"; CUB 3
       _dh_i=$((_dh_i + 1))
    done
    # note: cursor location was on top of ] after last shuttlecock print,
    #    but we then did CUB 3 as in every iteration of the prior loop
    #    so - to initialize properly for the next loop we need to more
    #    to the position each of its iterations will end at, and
    #    because its first act is CUB 5, we need to CUF 4 to get to ]+1
    CUF 4
    while [ "$_dh_i" -gt 0 ]; do     # should be same # of steps to get back
    # shift shuttle right
      CUB 5; printf '%s' "${_dh_l_cock} "; milli_sleep "${_dh_wait}"
       _dh_i=$((_dh_i - 1))
    done
    _dh_cycle=$((_dh_cycle + 1))
  done
  printf "\r"; message_n "shuttle zone: [                    ]"      # clear the zone aga
in (done)
  SCU   # show the cursor again after animation is over
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE
  printf "\n"

  #-----[ part 2a: DEC Graphics (Box Drawing) ]--------------------------------
  _dh_text="DEC Graphics (SO/SI Method)"
  # build command string for preview
  _dh_cmd="${_func_color}vt_init${_Boff}${_op_color};${_Boff}"
  _dh_cmd="${_dh_cmd} ${_func_color}box_shift${_Boff} ${_quote_color}\"${_dh_text}\"${_Bo
ff}"
  printf "\n"; message "${BYon}About to run:${Boff} ${_cmt_color}# High-fidelity SO/SI Bo
```

```
x${_Boff}\n ${_dh_cmd}"
  # logic execution
  vt_init; box_shift "${_dh_text}"
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  #-----[ part 2b: DEC graphics Explicit G0 Switch ]--------------------------
  _dh_text="Explicit G0 (ESCon/B)"
  # build command string for preview
  _dh_cmd="${_func_color}box_esca${_Boff} ${_quote_color}\"${_dh_text}\"${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff} ${_cmt_color}# High-fidelity ESCon/B
Box${_Boff}\n ${_dh_cmd}"
  # logic execution
  box_esca "${_dh_text}"
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  [ "$_dh_FAILED" = "$TRUE" ] && _dh_ret=1
  unset -v _dh_cmd _dh_FAILED _dh_i _dh_cycle _dh_cock _dh_wait
  return $_dh_ret
}

_dh_demo_environment() # (POSIX) demonstrate environment checks (isroot, checknotroot, ch
eckboot, etc)
{ # @usage _dh_demo_environment
  # @args none
  # @rules demonstrating environment and privilege checks
  _dh_ret=0
  _dh_FAILED=$FALSE

  separator "demonstrate_header-${BUILD}" "(environment & privileges)"

  #-----[ part 1: isroot - silent root privilege check ]-----------------------------
---------------------------------
  # construct the command preview string - isroot;  ... handle_result
  # part 1 - isroot; ...
  _dh_cmd="${_op_color}(${_Boff} ${_func_color}isroot${_Boff} ${_op_color});${_Boff} ...
"
  # [art 2 - handle_result
  _dh_cmd="${_dh_cmd}${_func_color}handle_result${_Boff}${_Boff} "
  printf "\n"; message "${BYon}About to run:${Boff} ${_cmt_color}# silent predicate check
${_Boff}\n ${_dh_cmd}"
  message_n "  Testing isroot (silent): "
  # logic execution
  isroot; _dh_result=$?
  if [ "$(id -u)" -eq 0 ]; then  ## we really are root (basis for test of isroot()
    [ "$_dh_result" -eq 0 ]
    handle_result $? "yes root and returned 0" "yes root but did not return 0"
  else
    # If we ARE NOT root, isroot should have returned 1 (Failure)
    [ "$_dh_result" -eq 1 ]
    handle_result $? "not root and returned 1" "not root but did not return 1"
  fi
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  #-----[ part 2: checkroot - gatekeeper root privilege check ]-----------------------
---------------------------------------
  # construct the command preview string - ( checkroot ); ... handle_result
  # part 1 - ( checkroot ); ...
  _dh_cmd="${_op_color}(${_Boff} ${_func_color}checkroot${_Boff} ${_op_color});${_Boff} .
.. "
  # part 2 - handle_result
  _dh_cmd="${_dh_cmd}${_func_color}handle_result${_Boff}${_Boff} "
  printf "\n"; message "${BYon}About to run:${Boff} ${_cmt_color}# root gatekeeper in sub
shell (exits if not root)${_Boff}\n ${_dh_cmd}"
  message_n "  Testing checkroot gatekeeper "
  # logic execution
  ( checkroot ) >/dev/null 2>&1 ; _dh_result=$?
  if isroot; then  ## we really are root (basis for test of checkroot()
    [ "$_dh_result" -eq 0 ]
    handle_result $? "yes root and returned 0" "yes root but did not return 0"
  else
```

```
    # If we ARE NOT root, isroot should have returned 1 (Failure)
    [ "$_dh_result" -eq 1 ]
    handle_result $? "not root and *exited* 1" "not root but did not exit 1"
  fi
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  #-----[ part 3: checknotroot - gatekeeper root privilege check ]----------------------
-----------------------------------------
  # construct the command preview string - ( checknotroot ); ... handle_result
  # part 1 - ( checknotroot ); ...
  _dh_cmd="${_op_color}(${_Boff} ${_func_color}checknotroot${_Boff} ${_op_color});${_Boff
} ... "
  # part 2 - handle_result
  _dh_cmd="${_dh_cmd}${_func_color}handle_result${_Boff}${_Boff} "
  printf "\n"; message "${BYon}About to run:${Boff} ${_cmt_color}# user gatekeeper in sub
shell (exits if root)${_Boff}\n ${_dh_cmd}"
  message_n "  Testing checknotroot gatekeeper: "
  # logic execution
  ( checknotroot ) >/dev/null 2>&1 ; _dh_result=$?
  if ! isroot; then  ## we really are not root (basis for test of checknotroot()
    [ "$_dh_result" -eq 0 ]
    handle_result $? "not root and returned 0" "not root but did not return 0"
  else
    # If we ARE root, checknotroot should have returned 1 (Failure)
    [ "$_dh_result" -eq 1 ]
    handle_result $? "yes root and *exited* 1" "yes root but did not exit 1"
  fi
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE
  # Note: In a real script, checkroot might exit; here we just observe.

  #-----[ part 4: checkboot - boot mount check ]----------------------------------------
------------------------------
  # construct the command preview string - ( checknotroot ); ... handle_result
  # part 1 - ( checkboot ); ...
  _dh_cmd="${_op_color}(${_Boff} ${_func_color}checkboot${_Boff} ${_op_color});${_Boff} .
.. "
  # part 2 - handle_result
  _dh_cmd="${_dh_cmd}${_func_color}handle_result${_Boff}${_Boff} "
  _dh_cmt="${_cmt_color}# verifies mounted status of boot/efi targets via bitmask exit st
atus${_Boff}\n"
  printf "\n"; message "${BYon}About to run:${Boff} ${_dh_cmt} ${_dh_cmd}"
  message_n "  Testing checkboot gatekeeper: "
  # logic execution - run checkboot; if it says fail, double-check
  _dh_OK_msg="All defined boot targets are mounted"
  _dh_FAIL_msg="One or more targets that should be is not mounted (Code: $_dh_result)"

  ( checkboot ) >/dev/null 2>&1 ; _dh_result=$?
  handle_result "$_dh_result" "${_dh_OK_msg}" "${_dh_FAIL_msg}"
  if [ ! "$_dh_result" -eq 0 ]; then
    # explain the failure code(s) using bitwise math
    _dh_i=0
    _dh_targets="/boot /efi /boot/efi"   # bitmask represents these in LSB->MSB order
    for _dh_tgt in $_dh_targets; do
      _dh_MOUNTPOINT=$FALSE
      # check if bit _dh_i is set in the result mask
      # (shift i bits right and check the LSB by bitwise AND w 1)
      if [ "$(( (_dh_result >> _dh_i) & 1 ))" -eq 1 ]; then
        # double-check checkboot's result using sys-util/util-linux's mountpoint
        # (checks whether a directory or file is a mountpoint. -q = silently)
        message_n "checkboot: ${BYon}${_dh_tgt}${Boff} should be but is ${BRon}not${Boff}
 mounted"
        mountpoint -q "${_dh_tgt}" && _dh_MOUNTPOINT=$TRUE
        if [ $_dh_MOUNTPOINT ] ; then
            printf '%s' "(${BRon}it is${Boff})"
            _dh_FAILED=$TRUE
            right_status 1
        else
            printf '%s' "(${BGn}it is not${Boff})"
            right_status $TRUE
        fi  # actually is mounted
```

```
      fi  # bit i is checkboot failure report
      _dh_i=$((_dh_i + 1))
    done  # for _dh_tgt
  fi  # check boot reports error?
  # Note: we are NOT double-checking if checkboot says all ok

  [ "$_dh_FAILED" ] && _dh_ret=1
  unset -v _dh_cmd _dh_cmt _dh_FAILED _dh_OK_msg _dh_FAIL_msg
  unset -v _dh_result _dh_i _dh_targets _dh_tgt _dh_MOUNTPOINT
  unset -v _dh_BOOT_REALITY
  return $_dh_ret
}

_dh_demo_widgets() # (POSIX) demonstrate separator, progress bar, countdown)
{ # @usage _dh_demo_widgets
  # @args none
  # @rules demonstrating established decorative and informative UI elements
  _dh_ret=0
  _dh_FAILED=$FALSE

  _dh_46="Donaldson Emmons Seward Seymour Panther Couchsachraga Santanoni"
  _dh_46="${_dh_46} Allen Cliff Redfield Colden Marshall Wright Algonquin"
  _dh_46="${_dh_46} Iroquois Street Nye Phelps TableTop Gray Skylight"
  _dh_46="${_dh_46} Marcy Haystack Basin BigSlide LowerWolfJaw UpperWolfJaw"
  _dh_46="${_dh_46} Armstrong Gothics Saddleback Giant RockyPeak Sawteeth"
  _dh_46="${_dh_46} Colvin Blake Dial Nippletop Dix Hough SouthDix"
  _dh_46="${_dh_46} GracePeak Macomb Porter Cascade Esther Whiteface"

  #-----[ part 1: separator - horizontal categorization ]-----------------------------
------------------------------
  # construct the command preview string - separator "DEMO" "(ui organization)"
  # line 1 part 1 - separator "DEMO"
  _dh_cmd="${_func_color}separator${_Boff} ${_quote_color}\"DEMO\"${_Boff} "
  # line 1 part 2 - "(ui organization)"
  _dh_cmd="${_dh_cmd}${_quote_color}\"(ui organization)\"${_Boff}${_op_color};${_Boff} "
  # line 1 part 3 - handle_result $?
  _dh_cmd="${_dh_cmd}${_func_color}handle_result${_Boff} ${_var_color}\$?${_Boff}\n"
  # line 2 - # horizontal rule with title
  _dh_cmd="${_dh_cmd} ${_cmt_color}# horizontal rule with title${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff}\n ${_dh_cmd}"
  # Logic execution
  separator "DEMO" "(ui organization)"
  [ "$?" -eq 0 ]
  handle_result $? "" "separator widget failed"
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE

  #-----[ part 2: progress_inline - task status ]-------------------------------------
------------------------------
  # construct the command preview string - for loop below
  # line 1 - _dh_i=1; HCU; _dh_offset=65;
  _dh_cmd="   ${_var_color}_dh_i${_Boff}${_op_color}=${_Boff}${_data_color}1${_Boff}${_op
_color};${_Boff} "
  _dh_cmd="${_dh_cmd}${_func_color}HCU${_Boff}${_op_color};${_Boff} "
  _dh_cmd="${_dh_cmd}${_var_color}_dh_offset${_Boff}${_op_color}=${_Boff}${_data_color}65
${_Boff}${_op_color};${_Boff}\n"
  # line 2 -    for _dh_peak in $_dh_46; do
  _dh_cmd="${_dh_cmd}   ${_ctl_color}for${_Boff} ${_var_color}_dh_peak${_Boff} ${_ctl_col
or}in${_Boff} "
  _dh_cmd="${_dh_cmd}${_var_color}\$_dh_46${_Boff}${_op_color};${_Boff} ${_ctl_color}do${
_Boff}\n"
  # line 3 -      progress_inline $_dh_i 47;
  _dh_cmd="${_dh_cmd}     ${_func_color}progress_inline${_Boff} ${_var_color}\$_dh_i${_Bo
ff} ${_data_color}47${_Boff}${_op_color};${_Boff}\n"
  # line 4 -      SCP; printf "\r";   ## note careful backslashes required (particularly
for \r -> \\\r)
  _dh_cmd="${_dh_cmd}     ${_func_color}SCP${_Boff}${_op_color};${_Boff} ${_func_color}pr
intf${_Boff} "
  _dh_cmd="${_dh_cmd}${_quote_color}\"\\\r\"${_Boff}${_op_color};${_Boff}\n"
  # line 5 -      CUF $_dh_offset; printf ...; EL; RCP;
  _dh_cmd="${_dh_cmd}     ${_func_color}CUF${_Boff} ${_var_color}\$_dh_offset${_Boff}${_o
```

```
p_color};${_Boff} "
  _dh_cmd="${_dh_cmd}${_func_color}printf${_Boff} ${_data_color}...${_Boff}${_op_color};$
{_Boff} "
  _dh_cmd="${_dh_cmd}${_func_color}EL${_Boff}${_op_color};${_Boff} ${_func_color}RCP${_Bo
ff}${_op_color};${_Boff}\n"
  # line 6 -    done; SCU;
  _dh_cmd="${_dh_cmd}   ${_ctl_color}done${_Boff}${_op_color};${_Boff} ${_func_color}SCU$
{_Boff}${_op_color};${_Boff}\n"
  # line 7 -    progress_inline 46 46; handle_result $?
  _dh_cmd="${_dh_cmd}   ${_func_color}progress_inline${_Boff} ${_data_color}46 46${_Boff}
${_op_color};${_Boff} "
  _dh_cmd="${_dh_cmd}${_func_color}handle_result${_Boff} ${_var_color}\$?${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff}\n ${_dh_cmd}"

  # Logic execution
  _dh_longest=$(get_longest ${_dh_46})
  _dh_offset=65   # empirically determined from progress_inline output (+10 after migrati
on for unk reason)
  _dh_i=1         # index from first peak, not 0
  _dh_wait=80     # millisecond loop wait time
  HCU  # hide the cursor until done tracking progress (distracting white block otherwise)
  for _dh_peak in $_dh_46; do
    # update progress bar
    # "lie" about progress target total to suppress final newline and get last peak name
on the right line
    progress_inline $_dh_i 47
    # save cursor position; return to left margin (absolute reference)
    SCP; printf "\r"
    # advance to position to the right of probress bar
    CUF $_dh_offset
    # print name of peak in fixed widht [] bounded "box",
    #   then erase the rest of the line to ensure it is clear, and restore cursor positio
n
    printf "${BCon}[${BMon}%-${_dh_longest}s${BCon}]${Boff}" "$_dh_peak"; EL; RCP
    _dh_i=$((_dh_i + 1))
    milli_sleep $_dh_wait
  done
  SCU  # cursor again (otherwise invisible and confusing on cli)
  progress_inline 46 46  # now that we finished w/o final newline, update the accurate %
progress
  [ "$?" -eq 0 ]; handle_result $? "reached 46 peaks" "progress_inline widget failed"
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE
  printf "\n"

  #-----[ part 3: sh_countdown - timed safety pause ]-----------------------------------
------------------------------
  # construct the command preview string -
  # line 1 -    sh_countdown 3 "Safety pause before deployment";
  _dh_cmd="   ${_func_color}sh_countdown${_Boff} ${_data_color}3${_Boff} "
  _dh_cmd="${_dh_cmd}${_quote_color}\"Safety pause before deployment\"${_Boff}${_op_color
};${_Boff}\n"
  # line 2 -    handle_result $?
  _dh_cmd="${_dh_cmd}   ${_func_color}handle_result${_Boff} ${_var_color}\$?${_Boff}"
  printf "\n"; message "${BYon}About to run:${Boff} ${_cmt_color}# interactive timer${_Bo
ff}\n ${_dh_cmd}"
  # Logic execution
  sh_countdown 3 "Safety pause before deployment"
  [ "$?" -eq 0 ]; handle_result $? "pause complete" "countdown widget failed"
  [ "$?" -eq 0 ] || _dh_FAILED=$TRUE
  printf "\n"

  [ "$_dh_FAILED" ] && _dh_ret=1
  unset -v _dh_cmd _dh_FAILED _dh_46 _dh_longest _dh_offset _dh_i _dh_wait _dh_peak
  return $_dh_ret
}

#_____________________________________________########################################----
-----------------------------------

demonstrate_header() # (POSIX) describe, demonstrate, and test capabilities of this heade
```

```
r
{
  # @usage demonstrate_header
  # @args none
  # @note each script header file in the script_header_joetoo family has a similarly name
d demonstrate_header function
  # @note e.g. demonstrate_header_unicode() in script_header_joetoo_unicode
  _dh_final_status=0   # clear all bits of this status "bitmask" variable

  # begin output with separator
  separator "demonstrate_header-${BUILD}" "(demonstration)"

  # execute sub-modules ( use a bitmask status "byte" to track overall status, indetail )
  # var|(1<<i) means: var = (binary shift a 1 i bits to the left) then bitwise OR that nu
mber with var
  # i.e. SET the ith most significant bit
  _dh_show_capabilities        || _dh_final_status=$(( _dh_final_status|(1<<0) ))
  _dh_demo_verbose             || _dh_final_status=$(( _dh_final_status|(1<<1) ))
  _dh_demo_versions            || _dh_final_status=$(( _dh_final_status|(1<<2) ))
  _dh_demo_numeric             || _dh_final_status=$(( _dh_final_status|(1<<3) ))
  _dh_demo_debug_utils         || _dh_final_status=$(( _dh_final_status|(1<<4) ))
  _dh_demo_boolean             || _dh_final_status=$(( _dh_final_status|(1<<5) ))
  _dh_demo_terminal_io         || _dh_final_status=$(( _dh_final_status|(1<<6) ))
  _dh_demo_environment         || _dh_final_status=$(( _dh_final_status|(1<<7) ))
  _dh_demo_widgets             || _dh_final_status=$(( _dh_final_status|(1<<8) ))

  #-----[ results summary ]-----------------------------------------------------------
----------------------
  # preview the command syntax
  # line 1 - message_n "${BCon}Finishing with status of summarization${Boff}"
  cmd="  ${_func_color}message_n${_Boff} ${_quote_color}\"\${BCon}Finishing with status o
f summarization\${Boff}\"${_Boff}\n"
  # line 2 - right_status $_dh_final_status
  cmd="${cmd}   ${_func_color}right_status${_Boff} ${_var_color}\$_dh_final_status${_Boff
}"
  # demonstrate message_n and right_status
  message "${BYon}About to run:${Boff}\n ${cmd}"
  message_n "${BCon}Finishing with status of summarization${Boff}"
  # also capture the final status in the bitmask
  _dh_result=$?; [ "$_dh_result" -eq 0 ] || _dh_final_status=$((_dh_final_status|(1<<9) )
)
  printf " (exit status: %b%s%b) --->" " $(status_color $_dh_final_status)" "${_dh_final_
status}" "$Boff"
  right_status $_dh_final_status
  printf "\n"

  #-----[ results summary ]-----------------------------------------------------------
----------------------
  separator "RESULTS" "(diagnostic summary)"

  # list functions in order from LSB (bit 0) to MSB
  _dh_tests="_dh_show_capabilities"               # bit 0
  _dh_tests="${_dh_tests} _dh_demo_verbose"       # bit 1
  _dh_tests="${_dh_tests} _dh_demo_versions"      # bit 2
  _dh_tests="${_dh_tests} _dh_demo_numeric"       # bit 3
  _dh_tests="${_dh_tests} _dh_demo_debug_utils"   # bit 4
  _dh_tests="${_dh_tests} _dh_demo_boolean"       # bit 5
  _dh_tests="${_dh_tests} _dh_demo_terminal_io"   # bit 6
  _dh_tests="${_dh_tests} _dh_demo_environment"   # bit 7
  _dh_tests="${_dh_tests} _dh_demo_widgets"       # bit 8
  _dh_tests="${_dh_tests} _dh_summarization"      # bit 9
  _dh_longest=$(get_longest $_dh_tests)
  _dh_fw=$((_dh_longest + 1))
  _dh_i=0
  for _dh_test_name in $_dh_tests; do
    # read the ith most significant bit
    # (binary shift binary value right by i bits) (binary bitwise AND with 1)
    # check if the ith bit is set: bit=$(( (mask >> i) & 1 ))
    _dh_bit=$(( (_dh_final_status >> _dh_i) & 1 ))
    # output Test i: testname
```

```
    printf " Test %02d: [${BMon}%-${_dh_fw}s${Boff}]" "$((_dh_i + 1))" "${_dh_test_name}"
    # render status using the handle_result library function
    # passing status bit, "passed" ok_msg, and "failed" error msg
    printf "(%b%1d%b)" " $(status_color $_dh_bit)" "$_dh_bit" "$Boff"
    handle_result $_dh_bit "passed" "failed"
    _dh_i=$((_dh_i + 1))
  done
  printf "\n"

  # clean up local variables for POSIX hygiene
  unset -v _sh_FUNCTION_REGEX _sh_METADATA_REGEX _sh_dir _sh_header_glob
  unset -v _sh_msg _sh_filespec _sh_PN _sh_fn_list _sh_fn_max_len
  unset -v _sh_fn_sep _sh_fn_col_width _sh_fn_col _sh_fn_indent
  unset -v _sh_label_list _sh_label_max_len _sh_label_sep
  unset -v _sh_label_col_width _sh_label_col _sh_label_indent
  unset -v _sh_line _sh_tmp _sh_fn_name _sh_meta _sh_tag _sh_payload
  unset -v _dh_result cmd _dh_tests _dh_longest
  unset -v  _dh_fw _dh_i _dh_test_name _dh_bit
  return $_dh_final_status
}
# metadata to FOLLOW summarize_header other output --
# @usage demonstrate_header
# @deps _dh_test_render message separator
# @rule iterates through core UI components (colors, cursor, math, messaging)
# @rule provides a "live" visual verification of POSIX terminal compatibility
# @ret exit 0 if all tests pass; 1 if any core logic fails
# @note use this to verify terminal behavior on new or minimal POSIX environments
# @note
# @note POSIX Argument Expansion Reference:
# @note "$@" (Quoted) -> THE GOLD STANDARD. Preserves arguments exactly as passed
# @cont Example: [ "arg one" "arg two" ] stays [ "arg one" "arg two" ]
# @cont Use for: d_do, wrappers, and passing arguments to other functions
# @note
# @note "$*" (Quoted) -> THE STRING BUILDER. Flattens all arguments into ONE string
# @cont Example: [ "arg one" "arg two" ] becomes [ "arg one arg two" ]
# @cont Use for: log_message, echo, and _strip_ansi (treats message as a single blob)
# @note
# @note $@ or $* (Unquoted) -> THE WORD SPLITTER. Dangerous/Rarely used
# @cont Triggers "Internal Field Separation" (IFS). Spaces break into new arguments
# @cont Example: [ "arg one" ] becomes [ "arg" "one" ]
# @cont Use for: for-loops where you explicitly want to iterate over words
# @rule Wrapper Rule: Always use "$@" to pass data to workers to prevent mangling
# @rule Logging Rule: Use "$*" when piping to sed/_strip_ansi for stream processing
```