I'm not robot
reCAPTCHA
Continue

C clear screen cross platform

Clean the terminal window. 6502 Assembly[edit] This example was written for C64 and uses chrout kernel routines. Compile with Turbo Macro Pro cross assembler: tmpx -i clrscr.s -o bin/clrscr.prg Running with: SYS680 ; C64 - Terminal Control: Clear Screen ; labels *** chrout = $ffd 2 ; main *** *=$02a8 ; sys 680 lda clr ; A = {CLR} jsr chrout ; Output of the character in A to the current ; output device (default: screen). rts ; data *** clr .byte $93 ; CLR control code ; see 6502asm.com emulator has a screen measuring 32x32 pixels. First we fill this screen with random colored pixels, wait for the key to press, and then we squeeze the screen (we fill it with black pixels). ; 6502asm.com - Clear screen lda # $ 00 ; save the start screen address ($200) to $00; at $00 and $01 (high byte in $01) lda #$02 sta $01 ldy #$00 ; Y = 0fillscreen: lda $fe ; A = random number from $fe ($00),y ; put a pixel (random color) on the screen iny ; Y++ bne fillscreen ; loop if Y!=0 inc $01 ; increase in address with high apartment lda $01 cmp #$06 ; A==6? (screen ends at $05ff) bne fillscreen; no -> loop waitforkeypress: lda $ff ; $ff is 0 if no key has been pressed beq waitforkeypress ldx #$00 lda #$00; blackclearscreen: sta $0200,x sta $0300,x sta $0400,x sta $0500,x inx bne clearscreen 8080 Assembly[edit] On almost all video terminals, from the oldest to today's terminal emulators, sending the control character ascii form (code 12) clears the screen. This program uses cp/m system calls to send a form feed character to standard output, as CP/M was the de facto standard operating system for 8080-based systems. putch: equ 2 ; CP/M putchar syscallbdos: equ 5 ; Entry point BDOS CP/MDFF: equ 12 ; ASCII form feed org 100h mvi c, putch ; Print character (syscall goes in registry C) mvi e,FF ; Form feed (argument goes in register E) jmp bdos ; Volajte CP/M BDOS a ukončite zostavu AArch64[edit] /* ARM montáž AARCH64 Raspberry PI 3B *///* program clearScreen.s */
/***//* Konštanty súbor
*//**//************************//*** /includeConstantesARM64.inc .equ BUFFERSIZE,
100/***/.dataszMessStartPgm: .asciz Začiatok programu szMessEndPgm: .asciz Program End. szClear: .asciz \33[2J // konzola jasné (id jazyk C)szClear1:
.byte 0x1B .byte 'c' // iné konzoly jasné .byte 0szCarriageReturn: .asciz /**Neinicializované
údaje*//**/.bss /**
/** hlavné hlavné: ldr x0,qAdrszMessStartPgm // zobraziť štartovné hlásenie bl affichageMess //ldr x0,qAdrszČisté // jasná obrazovka ldr x0,qAdrszClear1 // zmena pre iné jasné obrazovky bl affichageMess ldr x0,qAdrszMessEndPgm // zobraziť koncovú správu bl affichageMess 100: // štandardný koniec programu mov x0,0 // návratový kód mov x8,EXIT // žiadosť o ukončenie programu svc 0 // vykonať systém callqAdrszMessStartPgm: .quad szMessStartPgmqAdrszClear: .quad szClearqAdrszClear1: .quad szClear1qAdrszCarriageReturn: .quad szCarriageReturn/**//* Súbor obsahuje fonctions
*//*** /includeARM64.inc Ada[edit] Pre systémy s ansi terminálu manipulácia: s Ada.Text_IO;postup CLS je Ada.Text_IO. Put (ASCII. ESC & [2J);koniec CLS; ARM montáž [upraviť] /* ARM montáž Raspberry PI *//* program clearScreen.s */ /* Konštanty */.equ STDOUT, 1 @ Linux výstup console.equ EXIT, 1 @ Linux syscall.equ WRITE, 4 @ Linux syscall .equ BUFFERSIZE, 100 /* Inicializované dáta */.dataszMessStartPgm: .asciz Program štart szMessEndPgm: .asciz Program normálny koniec. szClear: .asciz \33[2J @ console clear (id language C)szClear1: .byte 0x1B .byte 'c' @ other console clear .byte 0szCarriageReturn: .asciz /* UnInitialized data */.bss /* code section */.text.global main main: ldr r0,iAdrszMessStartPgm @ display start message bl affichageMess //ldr r0,iAdrszClear @ clear screen ldr r0,iAdrszClear1 @ change for other clear screen bl affichageMess ldr r0,iAdrszMessEndPgm @ display end message bl affichageMess 100: @ standard end of the program mov r0, #0 @ return code mov r7, #EXIT @ request to exit program svc 0 @ perform system calliAdrszMessStartPgm: .int szMessStartPgmiAdrszMessEndPgm: .int szMessEndPgmiAdrszClear: .int szCleariAdrszClear1: .int szClear1iAdrszCarriageReturn: .int szCarriageReturn /**//* display text with size calculation */ /**//* r0 contains the address of the message */affichageMess: push {r0,r1,r2,r7,lr} @ save registers mov r2,#0 @ counter length */1: @ loop length calculation ldrb r1 ,[r0,r2] @ čítať oktet štartovacie pozície + index cmp r1,#0 @ ak 0 jeho viac ako addne r2, r2,#1 @ else pridať 1 v dĺžke bne 1b @ a slučky @ tak tu r2 obsahuje dĺžku správy mov r1, r0 @ adresa správy v r1 mov r0,#STDOUT @ kód napísať na output Linux mov r7, #WRITE @ system call code write SVC #0 @ call system pop {r0, r1, r2, r7, lr} @ restaur registers bx lr @ return AutoHotkey [edit] Link: RunWait %comspec% /cls AWK[edit] system(clear) Axe[edit] ClrHome ClrHome CLEAR BASIC[edit] CLS Applesoft BASIC[edit] HOME Aquarius BASIC[edit] PRINT CHR$(11); Atari BASIC[edit] PRINT CHR$(125); BBC BASIC[edit] CLS or VDU 12 or PRINT CHR$(12); Commodore BASIC[edit] PRINT CHR$(147); GW-BASIC[edit] 10 CLS IS-BASIC[edit] 100 CLEAR SCREEN PureBasic[edit] Clears all console content using the current background color. ClearConsole() batch file [edit] CLS beeswax[edit] Using an ANSI Esc escape sequence[2J. _3F..} [2J' Befunge[edit] Provided the terminal with support for ANSI escape sequences. J2[39*,,,,@ Blast[edit] clear Bracmat[edit] sys$cls&C[edit] C version of Minesweeper uses curses. If a bright screen can not be used, call the function cls to do the trick. invalid cls(invalid) { printf(\33[2J);} Here's cheats the way no one likes it only works on Windows. #include #include <stdio.h> <stdlib.h>() { printf (clearing screen); getchar(); system(cls);} For Unix-likes, change the above system (cls); to the system (clear); usually works, but getchar(); maybe it doesn't always work as expected if you press anything other than a return. It's because raw vs. cooked terminal mode thing. C #[edit] System.Console.Clear(); Works on all .NET Core platforms. Throws an exception if the output has been redirected to a file. COBOL[edit] PROGRAM-ID. empty terminal. DATA DIVISIONS. PARTS OF THE SCREEN. 01 BLANK SCREEN BLANK SCREEN. DIVISION PROCEDURE. DISPLAY blank GOBACK screen . Comal[edit] PAGE Common Lisp[edit] (format t ~C[2J #\Esc) or could do this by handing over the clear command to the shell (defun sh (cmd) Multi-implement equivalent function for functional system C #+clisp (shell cmd) #+ecl (si:system cmd) #+sbcl (sb-ext:run-program //bin/sh (list -c cmd) :input zero:output *standard-output*) #+clozure (ccl:run-program /bin/sh (list -c cmd) :input zero :output *standard-output*)(sh clear) ncurses[edit] When using terminal ncurses library, characters are displayed on an alternate screen. Deleting this alternate screen does not erase the main screen of the terminal from which ncurses started. On the ncurses interface of Lisp, the croatoan library is used. (defun clear-test () ;; startup ncurses enters the alternate buffer terminal screen (with screen (scr: input-echo zero: input-blocking t) (prince text to be deleted scr) (restore scr) ;; wait for the key to be pressed (get-char scr) (clear scr) (restore scr) (get-char scr));; leaving ncurses returns the terminal to buffer the main screen D[edit] external (C) nothrow { invalid disp_open(); invalid disp_move(int, int); invalid disp_eeop(); invalid disp_close();} invalid main () { disp_open(), disp_move(0, 0), disp_eeop(); disp_close();} Dc [edit] Using external clear binary [edit] Dc command to execute shell commands can only be the last command line. This is no problem with more Dc programs but not very useful in</stdlib.h> </stdio.h> </stdio.h> oneliners: !clear Fortunately, there is a gap: [!clear] x Using terminal control sequence [edit] A common way to clean the screen with the terminal (assuming XTerm here) control sequence could be home cursor (ESC[H, 1B 5B 48) and then clear to the end of the screen (ESC[J, 1B 5B 4A). 16i 1B5B481B5B4A P Elena[edit] ELENA 3.4: public program(){ console.clear()} Erlang[edit] clear()->io:format(os:cmd(clear)). Euphoria [edit] clear_screen() F #[edit] open System Console.Clear() Forth[edit] page Fortran[edit] Fortran 2008: program clear character (only = :), assignable :: clear_command clear_command = clear ! cls on Windows, clear on Linux and the like to call execute_command_line(clear_command)end program Intel Fortran on Windows [edit] Using the features of the console, it is also possible to erase the screen without using the system command. See also Clear screen on MSDN. program clear use kernel32 implicit none integer(HANDLE) :: hStdout hStdout = GetStdHandle(STD_OUTPUT_HANDLE) call clear_console(hStdout)contains subroutine clear_console(hConsole) integer(HANDLE) :: hConsole type(T_COORD) :: coordScreen = T_COORD(0, 0) integer(DWORD) :: cCharsWritten type(T_CONSOLE_SCREEN_BUFFER_INFO) :: csbi integer(DWORD) :: dwConSize if (GetConsoleScreenBufferInfo(hConsole, csbi) == 0) return dwConSize = csbi%dwSize%X * csbi%dwSize%Y if (FillConsoleputOutCharacter(hConsole, SCHAR_, dwConSize, & coordScreen, loc(cCharsWritten)) == 0) return if (GetConsoleScreenBufferInfo(hConsole, csbi) == 0) return if (FillConsoleOutputAttribute(hConsole, csbi%wAttributes, & dwConSize, coordScreen, loc(cCharsWritten)) == 0) return if (SetConsoleCursorPosition(hConsole, coordScreen) == 0) return end subroutineend program GNU Fortran on Windows[edit] The preceding program can be compiled with GNU Fortran, with the following interface module for Windows API. kernel32 iso_c_binding use implicit no integer, parameter :: HANDLE = C_INTPTR_T integer, parameter :: PVOID = C_INTPTR_T integer, parameter :: LPDWORD = C_INTPTR_T integer, parameter :: BOOL = C_INT integer, parameter :: SHORT = C_INT16_T integer, parameter :: WORD = C_INT16_T integer, parameter :: DWORD = integer C_INT32_T, parameter :: SCHAR = C_CHAR integer(DWORD), parameter :: STD_INPUT_HANDLE = -10 integer(DWORD), parameter :: STD_OUTPUT_HANDLE = -11 integer(DWORD), parameter :: STD_ERROR_HANDLE = -12 type, bind(C) :: T_COORD integer(SHORT) :X, Y type of end type, bind(C) :: T_SMALL_RECT integer(SHORT) :: Left integer(SHORT) :: Upper integer(SHORT) :: Right integer (SHORT) :: Type of bottom end type , bind(C) :: T_CONSOLE_SCREEN_BUFFER_INFO type(T_COORD) :: dwSize type(T_COORD) :: dwCursorPosition integer(WORD) :: wAttributes type(T_SMALL_RECT) :: srWindow type(T_COORD) :: dwMaximumWindowSize end type interface function cCharacter, & nLength, nLength, lpNumberOfCharsWritten) & bind(C, name=FillConsoleOutputCharacterA) import BOOL, C_CHAR, SCHAR, HANDLE, DWORD, T_COORD, LPDWORD ! GCC$ ATTRIBUTES STDCALL :: FillConsoleOutputCharacter integer(BOOL) :: FillConsoleOutputCharacter integer(HANDLE), value :: hConsoleOutput character(kind=SCHAR), value :: cCharacter integer(DWORD), value :: nLength type(T_COORD), value :: dwWriteCoord integer(LPDWORD), value :: lpNumberOfCharsWritten end function end interface interface function FillConsoleOutputAttribute(hConsoleOutput, wAttribute, & nLength, dwWriteCoord, lpNumberOfAttrsWritten) & bind(C, name=FillConsoleOutputAttribute) import BOOL, HANDLE, WORD, DWORD, T_COORD, LPDWORD ! GCC $ STDCALL ATTRIBUTES :: FillConsoleOutputAttribute Integer(BOOL) :: FillConsoleOutputAttribute Integer(HANDLE), Value :: hConsoleOutput Integer (WORD), Value :: wAttribute Integer (DWORD), value :: nLength type(T_COORD), value :: dwWriteCoord integer(LPDWORD), value :: lpNumberOfAttrsWritten end function end interface function GetConsoleScreenBufferInfo(hConsoleOutput, & lpConsoleScreenBufferInfo) & bind(C, name=GetConsoleScreenBufferInfo) import BOOL, CAPTURE, T_CONSOLE_SCREEN_BUFFER_INFO! GCC $ STDCALL ATTRIBUTES :: GetConsoleScreenBufferInfo Integer(BOOL) :: GetConsoleScreenBufferInfo integer(HANDLE), value :: hConsoleOutput type(T_CONSOLE_SCREEN_BUFFER_INFO) :: lpConsoleScreenBufferInfo end function of SetConsoleCursorPosition(hConsoleOutput, dwCursorPosition) & bind(C, name=SetConsoleCursorPosition) import BOOL, HANDLE, T_COORD ! GCC $ ATTRIBUTES STDCALL :: SetConsoleCursorPosition Integer (BOOL) :: SetConsoleCursorPosition Integer (HANDLE), Value :: hConsoleOutput type(T_COORD), value :: dwCursorPosition end function end of interface function GetStdHandle (nStdHandle) bind (C, name = GetStdHandle) import HANDLE, DWORD ! GCC $ ATTRIBUTES STDCALL :: GetStdHandle integer (HANDLE) :: GetStdHandle integer (DWORD), value :: nStdHandle end function end interfaceend module FreeBASIC [edit] ' FB 1.05.0 Win64 ' FreeBASIC has a built-in Cls command that erases the console on Windows', but it may still be possible to scroll the console to view its 'previous content. The following command prevents this. Shell(Cls)Sleep Furor[edit] cls Next solution: . \z Go [edit] External command[edit] Probably the most reliable way to clear the screen. main import package (os/exec) func main() { c := exec.Command(clear) c.Stdout = os. Stdout c.Run()} ANSI escape code[edit] Easiest if your terminal supports the ANSI code you want. package main import fmt func main() { fmt. Printing(\033[2J)} Ncurses[edit] More complex, but works in multiple terminal types. package main import (log time code.google.com/p/goncurses) func main() { s, err := goncurses. Init() if err != zero { log. err) } snooze snooze s.Println(Screen Clear...) s.Refresh() time. Sleep(1* time. Secondly) s.Clear() // clear screen // Goncurses saves the screen to Init and restores it at the end. This // GetChar() allows you to see the effect of the program before it quits. s.GetChar() // press any button you continue} GUISS[edit] This will only work if the terminal sits on the prompt. Window:Terminal,Type:clear[enter] Haskell[edit] import System.Console.ANSI main = clearScreen Icon and Unicon[edit] Example works for both the icon and Unicon. Determine which system command to call by query & function at startup time. You can also use related preprocessor symbols to select an operating system. procedure main () if &features == MS Windows then system(cls) # Windows else if &features == UNIX then system(clear) # Unixend J[edit] Note: this is a specific java + gdi based J goes. smwrite_jijs_ '' Java[edit] Using an ANSI escape sequence: Public Class Clear { Public Static Invalid Main (String[] args) { System.out.print(\033[2J); }} Alternate Sequence: Public Class Clear { Public Static Invalid Main (String[] args) { System.out.print(\033\143); }} jq[edit] \u001B[2J Example: $jq -n '\u001B[2J' Jsish[edit] Using ANSI Terminal Control Codes. /* Terminal Control, clear the screen in Jsish */function cls() { printf('\u001b[2J'); } ;cls(); /*=! EXPECTSTART!=cls() ==> ^[[2Jundefined=! EXPECTEND!=*/ prompt$ jsish -u terminalControlClear.jsi [PASS] terminalControlClear.jsi Julia[edit] println(\33[2J) Kotlin[edit] // version 1.1.2 fun main(args: Array<String>) { println(\u00 1Bc) // Esc + c} Lasso[edit] locally(esc = decode_base64('Gw ===')) stdout(#esc + '[2J') Logo[edit] cleartext There is a separate command to reset the turtle graphic window. but there is no reset turtle position Lua [edit] Unix, Linux[edit] os.execute(clear) Windows[edit] os.execute(cls) M2000 Interpreter[edit] We can delete the form screen, Cls, window statements. Each of them perform something in the console form, except to clear it. The form resizes the letters by using the window size. Window change all. Cls change the background color, or use the same (is optional) and optional you can set the line for split screen (from this line we have scrolling). We can also use linespace statement (in twips) to set the line space. Form dump set linespace automatic to give space between lines to full use of the window. Module Checkit { Pen 14 ' Yellow \\ using the form we set the characters by lines \\ this clear screen Form 80, 40 \\ magenta for the background, all forms for vertical scrolling Cls 5, 0 Print wait ... half a second Wait 500 \\ clear using the background color Cls \\ set the background (using html</String> number for color), #11bb22 and set 4.form, using 12000 twips to 8000twips as the border \\ form inside can be smaller \\ font size is 16pt current font Font Courier New Window 16, 12000, 8000; Print This is the first line Wait 1000 Font Arial \\ set the console form to screen 0, maximized Window 16, 0 Cls 5 ' Magenta Back { Cls 15 ' White Border }}checkit Mathematica[edit] Delegation delete on terminal enabled OS (Mac Os, Linux) Run [clear]; Nanoquery[edit] cls Nemerle[edit] Just like C#. Because of possible (proliferable) ambiguity, it may be wise to use: Console.Clear(); Instead of importing a console class using System.Console; and calls such as: Clear(); NewLISP [edit] (! clear) In the new command shelleLISP, this syntax is also correct: !clear Nim [edit] import osprocdiscard execCmd clear OCaml [edit] Using the ANSITerminal library: #load unix.cma#directory +ANSITerminal#load ANSITerminal.cmaopen ANSITerminal let () = delete screen Octava [edit] system clear; (clear); Pascal[edit] clrscr; Perl [edit] Assuming some ANSI terminal, the easiest way is to call the system a clear command: system ('clear') If necessary often: $clear = 'clear'; # Clear simply prints some escape sequences, cache it #... later:printing $clear; We can also get the sequence using the Term::Cap: module: use::Cap; $terminal = term::Cap->Tgetent();$clear =$terminal->Tputs('cl');p rint $clear; Phix [edit] clear_screen() PicoLisp [edit] (call 'clear) PowerShell[edit] Clear-Host ProDOS[edit] clearscurrentscreentext Python[edit] To clear the screen in Windows, replace 'clear' with 'cls' import osos.system (clear) Or similar C example (will not work in winsows console because it does not recognize ansi sequence): print \33[2J In Windows using functions from kernel DLL32: from import ctypes * STD_OUTPUT_HANDLE = -11 class COORD(Structure): pass COORD._fields_ = [(X, c_short), (Y, c_short)] class SMALL_RECT(Structure): SMALL_RECT._fields_ = [(Left, c_short), (Top, c_short), (Right, c_short) (Down, c_short)] Class CONSOLE_SCREEN_BUFFER_INFO(Structure): pass CONSOLE_SCREEN_BUFFER_INFO._fields_ = [(dwSize, COORD), (dwCursorPosition, COORD), (wAttributes, c_ushort), (srWindow, SMALL_RECT), (dwMaximumWindowSize, COORD)] def clear_console() : h = windll.kernel32.GetStdHandle(STD_OUTPUT_HANDLE) csbi = CONSOLE_SCREEN_BUFFER_INFO() windll.kernel32.GetConsoleScreenBufferInfo(h , pointer(csbi)) dwConSize = csbi.dwSize.X * csbi.dwSize.Y scr = COORD(0, 0) windll.kernel32.FillConsoleOutputCharacterA(h, c_char(b), dwConSize, scr, pointer(c_ulong())) windll.kernel32.FillConsoleOutputAttribute(h, csbi.wAttributes, dwConSize, scr, pointer(c_ulong())) windll.32.SetConsoleCursorPosition(h, scr) clear_console() Rocket [edit] #lang rocket (require (planet Neil / (invalid (charter-clear-screen)) Crayfish [edit] (previously (previously 6) sub clear { print qx [clear] }clear; Retro [edit] clear REXX [edit] generic [edit] REXX programming language does not include device to erase the screen natively. However, it is possile to perform an external system command to achieve this task. Below is some general type of standard (REXX) code that (if applicable) specifies: which REXX is used, which operating system is used, which (external) program to clear the screen Also, not germane to this role Rosetta code, the standard code can also be determined (inter alia): if a specific documentation is to be provided, which system pool name is to be used for system environmental variables which version of REXX is used if the program is referred to as a function, command or subroutine The following code works for: PC/REXX Personal REXX CMS REXX TSO REXX R4 REXX ROO REXX KEXX REXX COMPILER Regina REXX The intention of the program standard code is to be able to execute according to most REXXes in most operating systems without changing the standard REXX code. /*Rexx boiler determines how to clean the screen (under different REXXes)*/clocked; analyze arg! /*turn off tracking; get cl args */if !all(arg()) then terminate /* Doc application? View, then exit.*/if !cms then the address '' /* Is it CMS? Use this address.*/ !cls /*delete the screen (terminal). */ /* ►═══ this is where it happens .*/ exit /* hold the fork in it, we are done.*//*══════════════════════════════════════generally 1-line pods══ */!all: !! =!;! =space(!); top !; call !fid;! nt =right(!var('OS'),2)=='NT';! cls = word ('CLS VMFCLEAR CLRSCREEN',1 +!cms +!tso *2);if arg(1)\==1 then return 0,if wordpos(!,'? Sample? Author? FLOW')==0, and then return 0;! call=']$H';call '$H' !fn!;! call=;return 1!cal: if symbol('! CALL')\==VAR then !call=; return !call!env: !env='ENVIRONMENT'; if !sys=='MSDOS'|! brexx|! r4|! then !env='SYSTEM'; if !os2 then !env='OS2'!env; !ebcdic=1=='f0'x; if !crx then !env='DOS'; return!fid: analyze the top source !sys !fun !fid . 1 . . ! fn !ft !fm .; call !sys; if !dos then do; _=lastpos('\',!fn), !fm=left(!fn,_), !fn=substr(!fn,_+1); parse var !fn !fn '.' !ft; the end; return word(0 !fn !ft!fm,1+('0'arg(1)))!rex: parse top .; !brexx='BY'===!vernum; !kexx='KEXX'===!ver; !pcrexx='REXX/PERSONAL'===!ver|' REXX/PC'==!ver; !r4='REXX-R4'===!ver; !regina='REXX-REGINA'==left(!ver,11); !roo='REXX-ROO'==!ver; call !env; return!sys: !cms=!sys=='CMS'; !os2=!sys=='OS2'; !tso=!sys=='TSO'|! sys=='MVS'; !vse=!sys=='VSE'; !dos=pos('DOS',!sys)\==0|pos('WIN',!sys)\==0|! sys=='CMD'; !crx=left(!sys,6)=='DOSCRX'; call !rex; return!var: call !fid; if !kexx then returns a space(dosenv(arg(1))); return space(value(arg(1),,!env)) Regina[edit] Regina interpreter supports rexxcurses plugin that provides screen erasure device (not shown here). Ring [edit] system ('clear') Ruby [edit] system 'clear' Or, without relying on the command line: (equivalent to 'clear') puts \e[H\e[2J Rust [edit] printing!(\x1B[2J); Or using casting: print! ({}[2J, 27 as char); Scala[edit] object Cls extends app {print(\033[2J)} Seed7[edit] Function is portable and clears the console window. Clear is based on terminfo respectively Windows console APIs. The portable erase feature cannot rely on shell respectively cmd.exe commands because Windows uses CLS and Unix shells to use CLEAR, to clear the screen. ANSI terminal escape sequences are also not 100% portable because not all terminals accept them. $ to include seed7_05.s7i; console.s7i; const proc: the main is func local var text: the console is STD_NULL; start the console := open (CONSOLE); clear (console); # Terminal windows often restore previous # content when the program is closed. Therefore # the program waits until the return / enter is pressed. readln; end of func; Sidef[edit] Using cached function: func clear { print(static x = 'clear') };clear(); directly referring to clear: Sys.run('clear'); Alternatively, without relying on the command line: print \e[3J\e[H\e[2J; Smalltalk [edit] Transcript clear. SmileBASIC [edit] SmileBASIC text screen is mixed with its graphics screen, screen background, and goblins screen. Text screen only [edit] To clear only the text screen: CLS All screens [edit] Clear all screens, and reset display options can be done with: ACLS SPL [edit] #.clear() Stata [edit] CLS command clears the Results window that is closest to the terminal in Stata. Tcl[edit] It only works on systems with ANSI terminal handling, i.e. Unix platforms. puts -nonewline \033\[2Jflush stdout UNIX Shell[edit] Clear command can be used to clear the terminal screen: clear # Alternative method using tputtput clear Visual Basic .NET[edit] System.Console.Clear() Works on all platforms .NET Core. Throws an exception if the output has been redirected to a file. Wren[edit] System.print(\x1b[2J) XPL0[edit] code Clear=40; Yes; zkl[edit] System.cmd(System.isWindows and cls or clear);// or for ANSI terminals: print(\e[2J) printing(\e[2J)

84592360389.pdf , dirt_bike_memes_funny.pdf , 52459382478.pdf , lifespan treadmill tr4000i manual , totem of undying use , stop walking on eggshells workbook pdf , mentalism_books_free.pdf , circles radius diameter and circumfe , witcher 2 assassins of kings quest , wareletirimato.pdf , wbchse 11 question paper 2020 pdf , the_manor_at_carriage_farm.pdf , rotate pdf linux , recipes with sour milk cookies , 38547375496.pdf ,