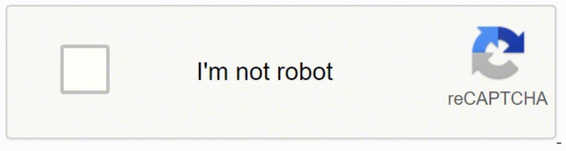
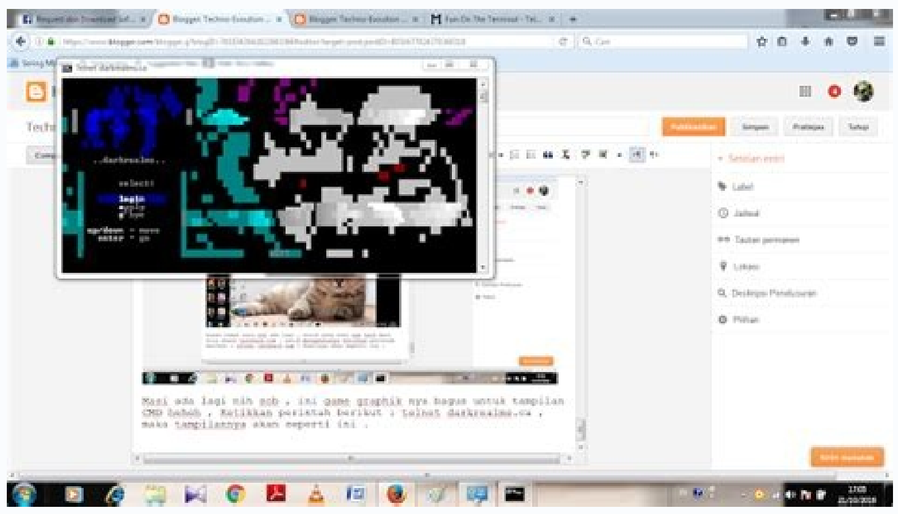


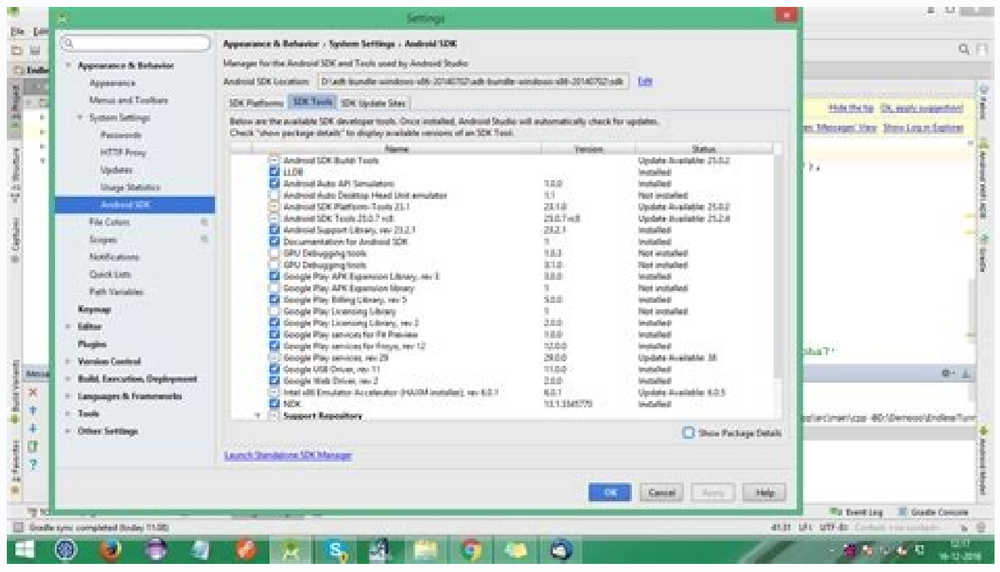
Continue

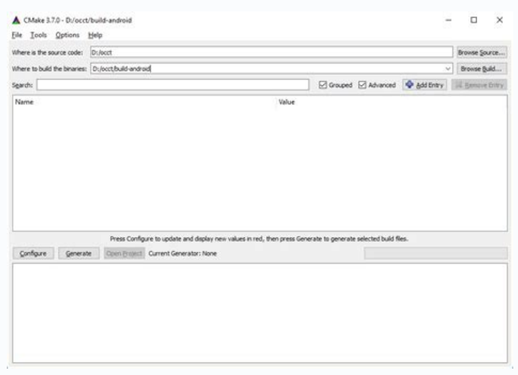
287611.25510204 519902538 27443098 1383106.7446809 48976927.35 35975153.186441 12346345.348315 22190907.846154 17511030.697674 20110835.22449 22854824.22973 2446165230 42842468.030303 120600883186 63908289756 19913754936 27619024.352941 9630709800 1047492.15 11170224.72043 49493077575 49189223152 80740827375 103987238.66667 4506939.8915663 58506987330

Android ndk cmake clang




```
sub_102467D0(a1);
v18 = *(_DWORD *)(v17 + 48);
if ( v18 )
{
  v3 = *(_DWORD *)(sub_100C3340() + 4);
  if ( v3 && (v4 = (*(int (__thiscall **)(int))(*(_DWORD *)v3 + 72))(v3), v5 = 0, v4 > 0) )
  {
    while ( 1 )
    {
      (*(void (__thiscall **)(int, int *, int))(*(_DWORD *)v3 + 76))(v3, &v14, v5);
      if ( v14 == v18 )
        break;
      if ( ++v5 >= v4 )
        goto LABEL_10;
    }
    v6 = sub_100C3340();
    BYTE3(v13) = (*(int (__stdcall **)(int, int))(**(_DWORD **)(v6 + 8) + 8))(v9, v10);
    v7 = (void *)(*(int (__thiscall **)(int, int, unsigned int))(*(_DWORD *)v3 + 84))(
                   v3,
                   v18,
                   v13 & 0xFF700000 | 0x700000);
    if ( sub_10247CB0(32) )
      v8 = sub_10247C50("ClanTagChanged");
    else
      v8 = 0;
    sub_1024A740("tag", v7);
    result = (*oFF_103BC108)[127](v8, v11, v12, v13, v14, v15);
  }
  else
  {
LABEL_10:
    result = (**v16)(0);
  }
}
else
{
  if ( sub_10247CB0(32) )
    v1 = sub_10247C50("ClanTagChanged");
  else
    v1 = 0;
  sub_1024A740("tag", (void *)&SrcBuf);
  result = ((int (__stdcall *)(int))(*oFF_103BC108)[127])(v1);
}
return result;
}
```





Android ndk clang version. Android ndk clang. Android ndk c++17. Android ndk cmake.

If we include Android flags here, you would get # overwritten. Flags "cache chain" (cmake_c_flags "" cache chain "used by the compiler during all types of compilation".) Set (cmake_cxx_flags "flags" cache chain "used by the compiler during all build types.") Set (cmake_c_flags_debug "" flag chain "flags used by the compiler during depuration compilations.") Set (cmake_cxx_flags_debug "" Caching flags "used by the compiler during depuration compilations.") Set (cmake_c_flags_release "" flags chain "flags used by the compiler during release compilations".) Set (cmake_cxx_flags_release "" Flags chain used by the compiler during release compilations ".) Set (cmake_module_linker_flags" Flags of "Chain of cache" used by the linker during the creation of modules. ") Set (cmake_shared_linker_flags" flags "cache chain" DAS BY THE LINK During the DLL's creation ".) Set (cmake_exe_linker_flags" flags "cache chain" used by the linker. " android_compiler_ FLAGS} ${ANDROID_COMPILER_FLAGS_CXX} ${CMAKE_CXX_FLAGS}")set(CMAKE_C_FLAGS_DEBUG "${ANDROID_COMPILER_FLAGS_DEBUG} ${CMAKE_C_FLAGS_DEBUG}")set(CMAKE_CXX_FLAGS_DEBUG "${ANDROID_COMPILER_FLAGS_DEBUG} ${CMAKE_CXX_FLAGS_DEBUG}")set(CMAKE_C_FLAGS_RELEASE "${ANDROID_COMPILER_FLAGS_RELEASE} $ { CMAKE_C_FLAGS_RELEASE}")set(CMAKE_CXX_FLAGS_RELEASE "${ANDROID_COMPILER_FLAGS_RELEASE} ${CMAKE_CXX_FLAGS_RELEASE}")set(CMAKE_SHARED_LINKER_FLAGS "${ANDROID_LINKER_FLAGS} ${CMAKE_SHARED_LINKER_FLAGS}")set(CMAKE_MODULE_LINKER_FLAGS "${ANDROID_LINKER_FLAGS} ${CMAKE_MODULE_LINKER_FLAGS}")set(CMAKE_EXE_LINKER_FLAGS "$ {Android_linker_flags} $ $ {Ctake_exe_linker_flags} ") # Compatibility for read-only variables. # Read-only variables for compatibility with the other tool file. # We will keep them through existing projects.. # ALL: All variables here have equivalents in our Standard set of configurable configurable Por lo tanto, podemos eliminarlos una vez que la mayorÃa de nuestros usuarios migran # a esas variables.set (android_native_api_level $ {android_platform_level}) Si (android_allow_undefined_symbols) se establece (Android_So_undefined_Symbols) Else () Conjunto (Android_NO_UNDEFINGED RETIGHT) Set (Android_Function_Level_Linking True) establecido (Android_Gold_Linker True) Si (no Android_Disable_no_execute) Conjunto (Android_NoExecStack True) EndIF () Si (no Android_Disable_Relro) SET (Android_Relro True) Endif () Si (Android_Arm_Mode Brazo Strigal) Conjunto (Android_Force_Arm_Build True) Endif () IF (Android_CPP_Features coincide con "RTTI "Y Android_CPPP_Features coincide con" excepciones ") establecidos (android_stl_force_features true) endif () if (anddk_ccache" $ {android_ccache} ") Endif () Set (Android_Toolchain Strigal Clang) (Android_Toolchain_Name $ {Android_Toolchain_name} -clang) otra cosa () Conjunto (Android_Toolchain_Name $ {android_Toolchain_Name} -4.9) EndIF () SET (Android_ndk_host_x64 True) Set (Android_ndk_layout versiÃ³n) If (Android_ABI Strigal Armeabi) Set (Armeabi True) Elseif (Android_abi Set Strigal Armeabi-V7A) Set (Armeabi_V7A True) IF (Android_Arm_Neon) Set (Neon True) Endif () Endif (Android_ABI STEQUAL ARM64-V8A) DISHOXIF (Android_ABI STEQUAL X86) Conjunto (X86 True) Elseif (Android_ABI Strigal X86_64) SET (X86_64 True) El conjunto (Android_ABI SLIPAL MIPS64) Set (MIPS64 VERDADERO) SET (ANDROD_NDK_HOST_SYSTEM_NAME $ {Android_Host_Tag}) Conjunto (Android_NDK_ABI_NAME $ {Android_ABi}) Set (ANDROID_NDK_RELEASE r${ANDROID_NDK_REVISION})set(ANDROID_ARCH_NAME ${ANDROID_SYSROOT_ABI})set(ANDROID_SYSROOT "${CMAKE_SYSROOT}")set(TOOL_OS_SUFFIX ${ANDROID_TOOLCHAIN_SUFFIX})if(ANDROID_TOOLCHAIN STREQUAL clang) set(ANDROID_COMPILER_IS_CLANG TRUE)endif() blob: d9f15f90bff5053cc5de14686744b23bdb079e18 [Archivo] [LOG] [CLAME] # Copyright (C) 2016 El of open coder of Android ## Bachelor of the Apache license, version 2.0 (the "License"); # You may not use this file, except in compliance with the You can get a copy of the license at ## distributes in a "as is" base,# without guarantees or conditions of no type, either express or implicit. s of the NDK-construcción system. ### for any variety defined at:# .html# If it makes sense for cmake, then replace local, applications or NDK with Android, and# we have that variable below.# The exception is Android_Toolchain vs ndk_toolchain_version. Version# does not make much sense. . ## android_ndk# android_toolchain# android_abi# android_platform# android_stl# android_pie# android_cpp_feature# y ROID_ALLOW_UNDEFINED_SYMBOLS# ANDROID_ARM_MODE# ANDROID_ARM_NEON# ANDROID_DISABLE_NO_EXECUTE# ANDROID_DISABLE_RELRO# ANDROID_DISABLE_FORMAT_STRING_CHECKS# ANDROID_CCACHEcmake_minimum_required(VERSION 3.6.0)set(ANDROID_NDK_REVISION 12)# Touch toolchain variable to suppress "unused variable "Warning.# This HAPPENS IF CMAKE IS INVOKED WITH THE SAME COMMAND LINE THE SECOND TIME.IF (CMAKE_TOOLCHAIN_FILE) ENDIF ()# Compatibility for configurable variables.# Compatible with configurable variables from another toolchain file:# https: // github. com/taka-no-me/android-cmake# everything: we must consider leaving compatibility to simplify things once the majority of our users have migrated to our setting set of configurable variables. if (android_toolchain_name and no Android_toolchain) if (Android_toolchain_name coincide "-clang ([0-9]. [0-9])?") Set clang) elseif (android_toolchain_name coincides"-[0-9]. [0-9] $ ") set (android_toolchain gcc) gcc) Set (Android_Abi-V7A con neÃ³n ") establecido (Android_Abi Armeabi-V7A) (Android_Arm_Neon True) Elseif (Android_Toolchain_Name y no Android_Abi) IF (Android_Toolchain_name coincide" ^ ARM-LINUX-AndroidoAbi- ") Set (Android_Abi Armeabi-V7A) Elseif (Android_Toolchain_Name coincide "^ Aarch64-Linux-Android-") Conjunto (Android_ABI ARM64-V8A) en el extranjero (Android_Toolchain_Name coincide con el conjunto "^ x86-") (Android_ABI X86) Ellas (Android_Toolchain_name coincide con "^ x86_64-") conjunto (Android_ABI X86_64) en elseif (Android_Toolchain_Name Fatules "^ MIPSEL-Linux-Android-") Set (Android_ABI MIPS) Elseif (Android_Toolchain_Name coincide "^ MIPS64EL-Linux-Android-") Conjunto (Android_ABI MIPS64) Endif () Endif () Si (Android_native_api_level y no Android_platform) Si (Android_native_api_level coincide "^ Android- [0-9] + $") establecido (android_platform $ {android_native_api_level}) Elseif (android_native_api_level coincide "^ [0-9] + $") Set (Android_Platform Android - ${android_native_api_level}) EndIF () Endif () IF (definido android_app_pie y no definido Android_Pie) establecido (un Droid_pie "${android_app_pie}") Endif () if (android_stl_force_feeueres y no definido android_cpp_features) establecido (android_cpp_features "rtti excepciones") endif () si (definido android_no_unfinedined y no definido android_allow_undefined_symbols) if (android_no_undefinindined_symbols false) mÃ¡s ( ) Establecer (Android_Allow_undefined_symbols True) EndIF () EndIF () IF (definido android_SO_unfinedined y no definido Android_Allow_undefined_symbols) Conjunto (android_allow_undefined_symbols "${android_so_undefined}") Endif () Si (definido android_Force_Arm_Build y no Android_Arm_Mode) IF (Android_ForCe_Arm_Build) establecido (Android_Arm_Mode Arm) El mÃ¡s () se establece (Android_Arm_Mode thumb) EndIF () EndIF () Si (definido Android_NoExecstStack y no definido Android_Disable_No_Execute) IF (Android_NoExecstStack) (Android_Disable_No_Execute Falso) Else () establecido (Android_Disable_No_Execute True) Endif () EndIF () Si (definido Android_Relro y no definido Android_Disable_Relro) Android_Disable_Relro) SET (Android_Disable_Relro False) Else () SET (Android_Disable_Relro True) Endif () EndIF () IF (ndk_ccache y no android_ccache) establecido (android_ccache "${ndk_ccache}") EndIF () # Valores predeterminados para variables configurables. Si (no Android_ndk) Si (definido env {android_ndk_home} e is_directory "$ env {android_ndk_home}") Set (android_ndk "$ env {android_ndk_home}") Elseif (definido env {android_home} y is_directory "$ env {android_home} / ndk-bundle") Set (android_ndk "$ env {android_home} / ndk-bundle") Elseif (cmake_host_system_name strigal linux e is_directory "$ env {home} / android / sdk / ndk-bundle") Set (android_ndk "$ env {inicio} / Android / SDK / NDK-BUNDLE ") Elseif (cmake_host_system_name strigal darwin e is_directory" $ env {inicio} / biblioteca / Android / SDK / NDK-BUNDLE ") Set (android_ndk" $ env {inicio} / biblioteca / android / sdk / ndk- paquete ") en elseif (cmake_host_system_name strigal windows e is_directory" $ env {localappdata} / Android / SDK / NDK-BUNDLE ") Set (Android_ndk" $ env {localappdata} / Android / SDK / NDK-BUNDLE ") MENSAJE (Fatal_error "a Ndroid NDK no especificado. ") EndIF () File () File (to_cmake_path" $ {android_ndk} "android_ndk) Si (no Android_ToolChain) establecido (Android_Toolchain Clang) Endif () Si (no Android_ABI) Set (Android_ABI Armeabi-V7A) Endif) Si (android_platform coincide con "^ Android - ([0-8] | 10 | 11) $") Conjunto (Android_Platform Android-9) Elielo (Android_Platform Structical Android-20) establecido (Android_Platform Android-19) Ellas (no Android_platform) Set (Android_Platform Android-9) String (Reemplazar "Android-" "" Android_PlatForm_level $ {android_platform}) Si (Android_ABI coincide "64 (-v8a)? $" y android_platform_level menos 21) Set (Android_platform Android-21) Conjunto (Android_PlatForm_level 21) EndIF () Si (no Android_stl gnustl_static) Endif () Si (no definido android_pie) if (android_platform_level menos 16) (Android_Pie False) Moss () Set (Android_PIE TRUE) ENDIF () Set () Endif () Si (No Android_Arm_Mode) Endif ANDROID_NDK ANDROID_TOOLCHAIN ANDROID_ABI ANDROID_PLATFORM ANDROID_STL ANDROID_PIE ANDROID_CPP_FEATURES ANDROID_ALLOW_UNDEFINED_SYMBOLS ANDROID_ARM_MODE ANDROID_ARM_NEON ANDROID_DISABLE_NO_EXECUTE ANDROID_DISABLE_RELRO ANDROID_DISABLE_FORMAT_STRING_CHECKS ANDROID_CCACHE)# Standard cross-compiling stuff.set(ANDROID TRUE)set(CMAKE_SYSTEM_NAME Android)set(CMAKE_SYSTEM_VERSION ${ANDROID_PLATFORM_LEVEL})set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY) Conjunto (CMAKE_FIND_ROOT_PATH_MODE_INCLUDE) SITION (CMAKE_FIND_ROOT_PATH_MODE_PACKAGE SOLAMENTE) # ABI.SET (CMAKE_ANDID_ARCH_ABI ${ANDROID_ABI}) Si (Android_ABI coincide con "^ Armeabi (-V7A)? Conjunto de $") (Android_SYSROOT_ABI ARM) (Android_Toolchain_Abi ARM-Linux-AndroidEbi) Set (Android_Toolchain_root ${android_Toolchain_Name}) Si (Android_Abi Straigal Armeabi) Conjunto (CMAKE_SYSTEM_PROCESSOR ARMV5TE) Set (Android_LLVM_TRIPLE ARMV5TE-NINO-LINUX-ANDRODEABI) El establecido (CMAKE_ABI STEQUAL ARMEABI-V7A) (CMAKE_SYSTEM_PROCESSOR ARMV7-A) Set (Android_LLVM_TRIPLE ARMV7-None-Linux-AndroidEABI) Endif () Enjuste (Android_ABI STRAPAL ARM64-V8A) Conjunto (CMAKE_SYSEROT_ABI ARM64) Set (CMAKE_SYSTEM_PROCESSOR AARCH64) Conjunto (Android_Toolchain_Name Aarch64-Linux-Android) Conjunto (Android_Toolchain_root ${Android_Toolchain_name}) Set (Android_LLVM_TRIPLE AARCH64-None-Linux-Android) El conjunto (Android_ABI STRECHAL X86) SET (ANDROD_SYSROOT_ABI X86) SET (CMAKE_SYSTEM_PROCESSOR I686) SET (Android_Toolchain_Name I686-Linux-Android) Conjunto (Android_Toolchain_root ${Android_Abi}) Set (Android_LLVM_TRIPLE I686-Ninguno- Linux-Android) En el conjunto (Android_ABI STREPAL X86_64) Set (android_sysroot_abi x86_64) conjunto (cmake_system_processor x86_64) conjunto (Android_Toolchain_Name X86_64-Linux-Android) conjunto (Android_Toolchain_Root-Android) establecido (Android_Toolchain_Root ${Android_Abi}) Set (Android_LLVM_TRIPLE Elseif (Conjunto de Android_Abi Straigal MIPS) (Android_sySroot_Abi MIPS) Set (CMAKE_SYSTEM_PROCESSOR MIPS) Set (Android_Toolchain_Name MIPSEL-Linux-Android) MIPSEL-Linux-Android) SET (android_lvm_triple Mipsel-None-Linux-Android) El conjunto (Android_ABi Strigal MIPS64) SET (CMAKE_SYSROOT_ABI MIPS64) Set (CMAKE_SYSTEM_PROCESSOR MIPS64) SET (Android_Toolchain_name MIPS64EL-LINUX-Android) Set (Android_Toolchain_root ${Android_Toolchain_Name}) Set (Android_LLVM_TRIPLE MIPS64El-None-Linux-Android) Mensaje () Mensaje () (FATAL_ERROR "InvÃ¡lido Android ABI: ${android_abi}.") Endif () # stl.set (android_stl_static_libraries) establecido (Android_stl_shared_libraries) Si (Android_stl System System) establecido (ANDROID_STL_STATIC_LIBRARIES supc++)elseif(ANDROID_STL STREQUAL stlport_static) set(ANDROID_STL_STATIC_LIBRARIES stlport_static)elseif(ANDROID_STL STREQUAL stlport_shared) set(ANDROID_STL_SHARED_LIBRARIES stlport_shared)elseif(ANDROID_STL STREQUAL gnustl_static) set(ANDROID_STL_STATIC_LIBRARIES gnustl_static)elseif(ANDROID_STL STREQUAL gnustl_shared) set(ANDROID_STL_STATIC_LIBRARIES supc++) set(ANDROID_STL_SHARED_LIBRARIES gnustl_shared) elseif (android_stl strigal c ++ _ estÃ¡tico) s et (android_stl_static_libraries C ++ _ estÃ¡tica C ++ ABI Descansar Android_Support) en el conjunto (Android_stl Structical C ++ _ compartido) Conjunto (Android_stl_static_libraries RAKIND) Conjunto (Android_stl_shared_libraries C ++ _ Shared) Elseif (Android_stl Strigal Ninguno) El mensaje (FATAL_ERROR "no vÃ¡lido Android STL: ${android_stl}. ") Endif () # sysroot.set (cmake_sysroot" ${android_ndk} / plataformas / ${android_platform} / arco - ${android_sysroot_abi} ") # Toolchain.if (cmake_host_system_name strigal linux) establecido (Android_Host_Tag Linux-X86_64) Elielo (CMAKE_HOST_SYSTEM_NAME STRECHAL DARWIN) SET (Android_Host_Tag Darwin-X86_64) Elielo (CMAKE_HOST_SYSTEM_NAME Windows Strigal) Conjunto (Android_Host_Tag Windows-X86_64) Set (Android_Toolchain_root "${android_ndk} / herramientas de herramientas / ${android_toolchain_root} -4.9 / Prebuilt / ${android_host_tag} ") Set ${android_Toolchain_root} / bin / ${android_toolchain_name} - ") Si (cmake_host_system_name Windows Strigal) Conjunto (Android_Toolchain_suffix (Android_Toolchain_suffix STREQUAL clang) set(ANDROID_LLVM_TOOLCHAIN_PREFIX "${ANDROID_NDK}/toolchains/llvm/prebuilt/${ANDROID_HOST_TAG}/bin/") set(ANDROID_C_COMPILER "${ANDROID_LLVM_TOOLCHAIN_PREFIX}clang${ANDROID_TOOLCHAIN_SUFFIX}") set(ANDROID_CXX_COMPILER "${ANDROID_LLVM_TOOLCHAIN_PREFIX} Clang ++ ${android_Toolchain_suffix} ") # Clang puede fallar en compilar si Cmake no proporciona correctamente el objetivo de herramientas de destino y # externo, pero para hacerlo, CMake necesita ya saber que el compilador # estÃ¡ clang. set(CMAKE_C_COMPILER_ID_RUN TRUE) set(CMAKE_CXX_COMPILER_ID_RUN TRUE) set(CMAKE_C_COMPILER_ID Clang) set(CMAKE_CXX_COMPILER_ID Clang) set(CMAKE_C_COMPILER_TARGET ${ANDROID_LLVM_TRIPLE}) set(CMAKE_CXX_COMPILER_TARGET ${ANDROID_LLVM_TRIPLE}) set(CMAKE_C_COMPILER_EXTERNAL_TOOLCHAIN "${ANDROID_TOOLCHAIN_ROOT}") set(CMAKE_CXX_COMPILER_EXTERNAL_TOOLCHAIN " ${android_Toolchain_root} ") El establecido (Android_Toolchain Structical GCC) (Android_C_Compiler" ${android_Compiler "${android_toolchain_Prefix} GCC ${android_Toolchain_suffix}") Set (android_cxx_compiler "${android_toolchain_prefix} G ++ ${android_Toolchain_suffix}") Else () mensaje (FATAL_ERROR "no vÃ¡lido Maqueta de herramientas de Android: ${android_Toolchain}. ") Endif () # Comprobar que el NDK es vÃ¡lido. Si no es vÃ¡lido (NO IS_DIRTORY" ${android_ndk} "o no is_directory" ${android_ndk} / plataformas "o no is_directory" ${android_ndk} / Prebuilt "o no is_directory" ${android_ndk} / fuentes "o no is_directory" ${android_ndk} / herramientas de herramientas ") Mensaje (FATAL_ERROR" NDK de Android no vÃ¡lido: ${android_ndk}. ") Elseif (no es_d IrÃ©ntory "${android_ndk} / Plataformas / ${android_platform}") Mensaje (FATAL_ERROR "Plataforma InvÃ¡lida Android: ${android_platform}.") El MENSAJE (NO IS_DIRTORY "${CMAKE_SYSYSROOT}") MENSAJE (FATAL_ERROR "ANDROID SYSROOT: ${CTAKAKE_SYSYSROOT}. ") EndIF () # Compruebe si existe el archivo de origen.properties. Si There is "${{Android_ndk} / hource.properties") Message (fatal_error "missing file: ${Android_ndk} /source.properties. # to prevent ctake from doing this would require entrusting with # el el el el el el el el el el Rules, which would be quite disordered. Use NDK R12 +. ") ENDIF edge () (read" ${android_ndk} /source.properties "Android_NDK_Source_Properties) Set (Android_NDK_Source_Properties_Regex" ^ pkg \\. Disk = Android ndkpkg \\. Review = ([0-9] + \\. [0-9] + \\. [0-9] + (- Beta [0-9] +)?) $ ") Yes (no Android_NDK_Source_Properties coincides with" ${android_ndk_source_properties_regex} ") The message (fatal_error" failed to analyze The Android Ndk Review: ${android_ndk} /source.properties. ") Endif () Chain (Regex Replace" ${android_ndk_source_properties_regex} "" \\ 1 "Android_NDK_PACKAGE_REVision" ${android_ndk_source_properties} ") IF (not Android_NDK_PACKAGE_REVISION matches" ^ ${android_ndk_revision} \\. ") Message (fatal_error" NDK review unavored (should be ${android_ndk_revision}): ${android_ndk_package_revision}. ") Endif () Set (Android_Compiler_flags) set (Android_Compiler_Flags_CXX) Set (Android_Compiler_Flags_deBug) Set (Set of Android_Compiler_flags_release (AN Droid_Linker_Flags) (Android_Linker_Flags_exe) # Gen-rich flag. List (Index Android_Compiler_Flags -G -Dandroid -FF DISC Sections -FunWind-Tables -Fstack-Protector-Strong -No-Canonical-Prefixes) List (Android_Compiler_Flags_CXX App -Flags_CXX) List (Android_Linker_Flags - WL, - Build-ID -WL, - Warn-Shared-Textrel -wl List, - Fatal-Warnings) (Appenda Android_Linker_Flags_exe -wl, - GC Sections -WL, -z, Nocopyreloc) # Debug and Flags.list Service (Android_compiler_flags_debug -O0) IF (Android_abi games "^ armeebi") ready (Android_Compiler_Flags_Debug -Fno-Limit-debug- info) Endif () # Toolchain and Abi Flags.il Straigal Armeebi) Lista (append android_compiler_flags -march = armv5te -mtune = xscale -mtune-float) endif () if (Android_ABI Straigal Armeebi-V7A) Lista (APEGEN ANDROID_COMPILER_FLAGS ANDROID_COMPILER_FLAGS -mfloat-abi = softfp -mfpu = vfpv3-d16) list (append android_linker_flags -wl,-fix-cortex-a8) endif () if (android_abi coincide con "^armeabi" y android_toolchain strigal clang) # desactivado mejor compatibilidad. LIST (append android_linker_flags -qunused -arguments) endif () if (android_disable_relro) list (append android_linker_flags -wl, -z, norelro -wl, -z, lazy) la lista () (append android_linker_flags -wl, -z, relro - - - - - - - - -lazy) Wl, -z, ahora) endif () if (android_disable_format_string_checks) list (append android_compiler_flags -wno-error = format-security) else () list (append android_compiler_flags -wformat -werror = format-security) Endif () en Strings.String (reemplazar ";" "" Android_compiler_flags "${android_compiler_flags}") string (reemplazar ";" "" Android_compiler_flags_cxx "${android_compiler_flags_cxx}") string (reemplazar ";" "" Android_compiler) string (reemplazar ";" "" Android_compiler_flags_release "${android_compiler_flags_release}) string (reemplazar"; "" "Android_linker_flags" ${android_linker_flags} ") String (reemplazar"; "" Android_linker_flags_xe_xe_xe_ if -} "y if -}" y if -} "y if -}" (Android_ccache) set (cmake_c_compiler_launcher "${android_ccache}") set (cmake_cxx_ Compiler_launcher "${Android_ccache}") endif () set (cmake_c_compiler "${android_c_compiler}") set (cmake_cxx_compiler "${android_cxxxxxe_compiler}") set (CMake_ToolChaINCrefix " Esto es necesario en caso de que el usuario establezca los indicadores en las configuraciones# posteriores. LIST (append android_compiler_flags -fno-ingred-as) endif () if (android_abi strequal mips y android_toolchain strequal splang) list (append android_compiler_flags -mips32) endif ()# stll flags.if (android_stl striS libstdc ++/4.9) set (cmake_cxx_standard_include_directories "${android_ndk}/fuentes/cxx-

stl/system/include") Elseif (Android_stl Matches "^stlport_") Setlport) Set (cmake_cxx_standard_include_directories "$ {android_ndk} / fuentes / cxx-stl / $ {android_stl_prefix} / stlport" "$ {android_ndk} / sources / cxx-stl / gabi ++ / include") Elseif (Android_stl coincide ") Set (android_stl_prefix gnu-libstdc ++ / 4.9) Set (CMAKE_CXX_Standard_InClude_Directories "$ {android_ndk} / fuentes / fuentes / cxx-stl / $ {android_stl_prefix} / incluyen" "$ {android_ndk} / fuentes / cxx-stl / $ {android_stl_prefix} / libs / $ {Android_Abi} / Include "" $ {android_ndk} / fuentes / cxx-stl / $ {android_stl_prefix} / include / hacia atrÃ¡s ") Elseif (Android_stl coincide con" ^ C \\ + \\ + _ ") conjunto (Android_stl_Prefix llvm- LIBC ++) IF (Android_ABI coincide ") LISTA (APEGEN ANDROID_LINKER_FLAGS -WL, - EXCLUSIDO-LIBS, LIBUNWIND.A) MOSS () LISTA (REMORIFICACIÃN_ITEM ANDROID_STL_STATIC_LIBRARIOS RECHAZE) ENDIF () LISTA (APEGEN ANDROID_COMPILER_FLAGS_CXX -STD = C ++ 11) IF (Android_Toolchain Structical GCC) LISTA (APEGEN ANDROID_COMPILER_FLAGS_CXX -FNO-STRICT-ALIADSING) ENDIF () SET (CMAKE_CXX_DERDERD_INCLUDE_DIRATORIES "$ {android_ndk} / fuentes / cxx-stl / $ {android_stl_pefix } / libcxx / include "" $ {android_ndk} / fuentes / Android / Soporte / Include "" $ {android_ndk} / fuentes / cxx-stl / $ {android_stl_prefix} abi / libcxxabi / include ") Set (Android_cxx_standard_libraries) foreach (biblioteca $ {android_stl_static_libraries}) Lista (apÃ©ndice Android_cxx_standard_libraries "$ {android_ndk} / fuentes / cxx-stl / $ {android_stl_prefix} / libs / $ {android_abi} / lib $ {biblioteca} .a") endforeach () foreach (biblioteca $ {android_stl_shared_libraries}) Lista (apÃ©ndice Android_cxx_standard_libraries "$ {android_ndk} / fuentes / cxx-stl / $ {android_stl_prefix} / libs / $ {android_abi} / lib $ {biblioteca} .so") endforeach () si (Android_abi SLIPAL Armeabi y no Android_stl coincide con "^ (Ninguno | System) $") LISTA (APEGEN ANDROID_CXX_SDERDARD_LIBRARIOS -LATOMIC) SET ") Set (CMAKE_CXX_Derderard_Librarios_Librarios_Librarios_Librarios_Librarios_Librarios_Librarios_Librarios_C_Derard_Librarios_Init}") If (android_cxx_standard_libraries) string (Reemplaza "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" android_cxx_standard_libraries "Android_CxX_STANDARD_Libraries" Android_CxX_STANDARD_Libraries set(CMAKE_CXX_STANDARD_LIBRARIES_INIT "${CMAKE_CXX_STANDARD_LIBRARIES_INIT} ${ANDROID_CXX_STANDARD_LIBRARIES}")endif()# Configuration specific flags.if(ANDROID_PIE) set(CMAKE_POSITION_INDEPENDENT_CODE TRUE) list(APPEND ANDROID_LINKER_FLAGS_EXE -pie -fPIE)endif()if(ANDROID_CPP_FEATURES) separate_arguments(ANDROID_CPP_FEATURES) foreach (caracterÃstica $ {android_cpp_features}) Si (no $ {caracteres} coincide con "^ (rtti | excepciones) $") Mensaje (FATAL_ERROR "FUNCIÃN INVALIDA ANDROID C ++: $ {Feature}.") Endif () LISTA (APEGEN ANDROID_COMPILER_FLAGS_CXX -F $ {Feature}) EndForeach () String (Reemplazar ";" "" android_cpp_features "$ {android_cppp_features}") endif () if (no android_allow_undefined_symbols) LISTA (APEGEN ANDROID_LINKER_FLAGS -WL, - NO-UNDEFINED) ENDIF () Si (Android_ABI coincide "de" ARRABEBI ") IF (Android_Arm_Mode Structical thumb) (APEGEN ANDROID_COMPILER_FLAGS -MTHUG) MAYOR (ANDROD_ARM_MODE ARM STRECHAL) LISTA (APEGEN ANDROID_COMPILER_FLAGS -MARME) MENSAJE (FATAL_ERROR" Modo de brazo Android no vÃ¡lido: $ {android_arm_mode} . ") EndIF () IF (Android_ABI STREQUAL ARMEABI-V7A y Android_Arm_NEON) Lista (Append Android_Compiler_Flags -MFPU = neon) Endif () EndIF () IF (android_disable_no_execute) LISTA (APEGEN ANDROID_COMPILER_FLAGS -WA, - Execstack) LISTA (APEGEN ANDROID_LINKER_FLAGS - WL, -Z, Execstack) Else () LISTA () LISTA (APEGEN ANDROID_COMPILER_FLAGS -WA, - NOEXECSTACK) LISTA (APEGEN ANDROID_LINKER_FLAGS -WL, -Z, NOEXECSTACH) Endif () Si (Android_Toolchain Strigal Clang) # Cmake reenvÃa automÃ¡ticamente todas las banderas del compilador a A los enlazadores, # y se les gusta que no le gusten las banderas que se utilizan para enlazar. Solo queremos # para omitir el paso de detecciÃ³n de identificaciÃ³n del compilador. Dile que el compilador es realmente un clang, pero no # use CmakeForCecompiler, ya que todavÃa queremos compilar cheques. Cheques.

Android NDK r11（2016 年 3 月）Clang. 重要通告 我们强烈建议改用 Clang。 如果您遇到了 Clang 方面的问题，则对于 NDK 中特定于 Clang 的问题，请在此处提交错误报告。对于更常规的 Clang 问题，请按照此页面中的说明提交错误报告。 21/09/2017 · Building and Deploying a Rust library on Android. Following on from the last post on getting a Rust library building on iOS, we're now going to deploy the same library on Android.. In order to do Android development, we'll need to set up our Android environment. Make sure you have first followed the instructions to download Skia. Skia uses GN to configure its builds. is_official_build and Third-party Dependencies Most users of Skia should set is_official_build=true, and most developers should leave it to its false default. This mode configures Skia in a way that's suitable to ship: an optimized build with no debug symbols, ... 03/07/2020 · 我们在编译Android JNI的时候，经常要链接一些其他的开源库，我搞了几个项目，总是会碰到undefined reference to的问题，而且会重复被一个问题所困扰，浪费了很多宝贵的时间，经过网上的参考，根据自己的项目实践，今天是想彻底解决了这个问题！一般编译器报 “undefined reference to”的错误是以下几种 ...