PARTE I · GUÍA PARA PRINCIPIANTES

Breeze

Manual de usuario · empezar desde cero

Esta guía está pensada para quien no conoce Breeze. La CLI ya está instalada; no necesitas compilarla ni instalarla de nuevo. Primero crearás y ejecutarás un proyecto pequeño. Después aprenderás a modificarlo y, cuando lo básico funcione, a añadir módulos y dependencias.

Antes de empezar

• Tener instalada la CLI Breeze y poder ejecutar `breeze --help` en una terminal.

• Tener Git, CMake 3.25 o superior y un compilador de C++20 compatible con tu sistema.

• Tener acceso a Internet para obtener el SDK y descargar herramientas o dependencias cuando corresponda.

Si `breeze --help` muestra la ayuda, la CLI está disponible. Si el comando no se reconoce, resuelve primero la instalación o el PATH de la CLI; eso queda fuera del flujo normal de esta guía.

Configurar la fuente del SDK una sola vez

Breeze necesita saber desde dónde obtener las versiones del SDK que usarán los proyectos. Usa el repositorio oficial del SDK de AcruxDevelopment como fuente:

```
breeze sdk-source --set "https://github.com/AcruxDevelopment/BreezeSDK"
```

Este comando configura la fuente que usará Breeze; no reemplaza la instalación de la CLI. Si ya instalaste o clonaste el SDK siguiendo las instrucciones de ese repositorio, conserva esa instalación. La fuente debe apuntar al repositorio indicado, no a una carpeta interna como `breeze/` ni a un archivo ZIP.

PARTE I · GUÍA PARA PRINCIPIANTES

Crear tu primer proyecto

Un proyecto es la carpeta que contiene tu código y la configuración necesaria para compilarlo. En los siguientes ejemplos, el proyecto se llamará Aurora. Puedes usar otro nombre, pero mantén el mismo nombre en los comandos posteriores.

1. Elegir dónde guardar los proyectos

mkdir C:\dev\projects
cd C:\dev\projects
En Linux o macOS, crea una carpeta equivalente en tu directorio personal y entra en ella. Ejecuta los siguientes comandos desde la carpeta donde quieras que aparezca el proyecto.

2. Crear el proyecto

breeze project new Aurora --sdk 1.0.0
Breeze crea la carpeta del proyecto usando la versión del SDK indicada con --sdk. En este ejemplo, 1.0.0 es ilustrativa: utiliza una versión publicada y disponible en la fuente que configuraste. Si la CLI informa que no encuentra esa versión, revisa las versiones disponibles en el repositorio del SDK y sustituye el número.

3. Entrar en el proyecto y preparar las herramientas

cd Aurora
breeze setup
`breeze setup` comprueba las herramientas necesarias para desarrollar y puede ofrecer instalar las que falten. Lee las preguntas antes de aceptar. No es necesario ejecutar la preparación cada vez que abres el proyecto; normalmente se hace al configurarlo por primera vez o cuando cambias de equipo o de herramientas.

¿Cómo sé que estoy en la carpeta correcta?

Estás en la raíz del proyecto cuando ves archivos como `CMakeLists.txt` y las carpetas `breeze/`, `project/` y `source/`. Los comandos de proyecto deben ejecutarse desde esa carpeta o desde una subcarpeta de ella.

PARTE I · GUÍA PARA PRINCIPIANTES

Compilar y ejecutar

Compilar significa convertir el código fuente en un programa ejecutable. Ejecutar significa iniciar ese programa para comprobar lo que hace. En el uso diario, estos son los dos comandos más importantes.

1. Compilar

breeze build
Breeze configura el sistema de compilación y compila los módulos habilitados. La primera compilación puede tardar más que las siguientes porque se descargan o preparan herramientas y dependencias.

2. Ejecutar

breeze run
Un proyecto recién creado suele incluir un ejecutable de ejemplo. Si hay más de un ejecutable, lista los módulos y especifica el ID del que quieras iniciar:

breeze ls
breeze run <id_del_ejecutable>
Reemplaza `<id_del_ejecutable>` por el ID que muestre `breeze ls`; no escribas los signos `<` y `>` literalmente.

3. Elegir plataforma y compilador

Un preset es una combinación de sistema operativo, arquitectura y compilador. Si Breeze no selecciona automáticamente el que necesitas, puedes indicarlo al compilar y ejecutar:

breeze build -p windows-x64-msvc
breeze run -p windows-x64-msvc
En Windows, `windows-x64-msvc` corresponde a Windows x64 con MSVC. En Linux puedes usar, por ejemplo, `linux-x64-gcc` o `linux-x64-clang`. Consulta los presets disponibles para tu sistema si uno no funciona.

Dónde queda el programa

Los archivos generados se guardan normalmente bajo `out/build/`, organizados por preset. Los ejecutables suelen estar en `bin/` y las bibliotecas en `lib/`. No edites esos archivos generados: cambia el código fuente y vuelve a compilar.

PARTE I · GUÍA PARA PRINCIPIANTES

Modificar el código y repetir el ciclo

Ahora harás un cambio pequeño para entender cómo se desarrolla un proyecto. La idea es repetir siempre el mismo ciclo: editar, guardar, compilar, ejecutar y comprobar el resultado.

1. Encontrar el código de ejemplo

Abre la carpeta del proyecto con tu editor de código. Dentro de `source/` encontrarás las carpetas de los módulos. Busca el `main.cpp` del ejecutable inicial y ábrelo.

```
#include <iostream>

int main()
{
   std::cout << "Hola desde Aurora\n";
   return 0;
}
```

`main` es el punto de entrada del programa. `std::cout` escribe texto en la consola. Si el archivo contiene otro mensaje, puedes cambiarlo por el ejemplo anterior; no reemplaces el contenido de otros archivos.

2. Compilar y comprobar el cambio

```
breeze build
breeze run
```

Si ves el nuevo mensaje, ya hiciste tu primer cambio correctamente. Si no aparece, confirma que editaste el `main.cpp` del ejecutable que estás ejecutando y que la compilación terminó sin errores.

Consejos para el trabajo diario

- Haz cambios pequeños y compila con frecuencia: así es más fácil encontrar errores.
- Lee los mensajes del compilador; suelen indicar el archivo y la línea que debes revisar.
- Si no recuerdas una opción, usa `breeze --help` o `breeze <comando> --help`.
- No modifiques manualmente los archivos generados dentro de `out/`; se pueden regenerar.

PARTE I · GUÍA PARA PRINCIPIANTES

Entender la estructura básica

No hace falta conocer todos los archivos para empezar. Al principio, céntrate en `source/`, que contiene los módulos y el código que vas a escribir.

```
Aurora/
├─ CMakeLists.txt    configuración principal
├─ breeze/           archivos y herramientas del SDK
├─ project/          ajustes propios del proyecto
├─ source/           código fuente organizado por módulos
├─ vendor/           dependencias incorporadas al repositorio
└─ out/              archivos generados al compilar
```

La carpeta `breeze/` pertenece al SDK. Evita editarla para personalizar tu aplicación, porque una actualización puede reemplazarla. Escribe tu código en `source/`, ajusta la configuración del proyecto en `project/` y guarda allí las dependencias que correspondan.

Consultar y cambiar los datos del proyecto

Breeze permite consultar o modificar metadatos como el nombre, el ID y la versión. Los comandos sin `--set` muestran el valor actual; al añadir `--set`, estableces uno nuevo:

```
breeze project name
breeze project name --set "Aurora Tools"
breeze project version --set 0.2.0
breeze project id --set aurora
```

Cambia estos valores solo cuando tengas claro cómo se usan en tus herramientas y scripts. El nombre visible y el ID son conceptos distintos: el ID se utiliza para identificar el proyecto o sus módulos en comandos.

PARTE I · GUÍA PARA PRINCIPIANTES

Añadir un módulo cuando lo necesites

Cuando el programa crece, puedes separar partes del código en módulos. Un módulo puede ser un ejecutable o una biblioteca. No necesitas crear módulos adicionales para un primer programa; esta sección es el siguiente paso, cuando quieras reutilizar código o dividir responsabilidades.

1. Crear una biblioteca estática

breeze add lib Math --static
Breeze crea un módulo de biblioteca con un ID que puedes consultar mediante `breeze ls`. También genera archivos para la interfaz pública y la implementación. La cabecera pública declara qué funciones se pueden utilizar; los archivos fuente contienen el código que las implementa.

2. Crear un ejecutable que use esa biblioteca

breeze add exec Game --deps math_lib
El ID exacto de la biblioteca debe coincidir con el que aparece en `breeze ls`; en este ejemplo se usa `math_lib`. Abre el `main.cpp` del módulo Game e incluye la cabecera pública de Math. Después llama a una función que realmente esté declarada e implementada en esa biblioteca.

3. Compilar y ejecutar

breeze ls
breeze build
breeze run game_exe
Si falla la compilación, confirma que la biblioteca existe, que el ID de la dependencia es correcto y que el nombre de la cabecera coincide con el archivo generado. Los nombres de funciones de ejemplo pueden variar según la plantilla y el contenido que hayas editado.

PARTE I · GUÍA PARA PRINCIPIANTES

Dependencias y solución de problemas

Una dependencia indica que un módulo necesita otro módulo o una biblioteca para compilar o enlazarse. En Breeze puedes declararla al crear un módulo con `--deps`, o en la configuración CMake del módulo. Las dependencias deben existir y estar configuradas; escribir su nombre no instala automáticamente una biblioteca externa.

Consultar dependencias

breeze deps of game_exe
breeze deps on math_lib
El primer comando permite consultar qué necesita un módulo; el segundo ayuda a ver qué módulos dependen de una biblioteca. Sustituye los IDs por los que muestre tu proyecto.

Errores frecuentes

• “No es un proyecto Breeze”: entra en la carpeta que contiene el `CMakeLists.txt` principal y vuelve a ejecutar el comando.
• No encuentra el SDK: comprueba la fuente con `breeze sdk-source` y configúrala con `breeze sdk-source --set "https://github.com/AcruxDevelopment/BreezeSDK"`.
• No encuentra un módulo: ejecuta `breeze ls` y copia el ID exacto.
• No compila: lee el primer error del compilador, revisa el archivo y la línea indicados, corrige y vuelve a ejecutar `breeze build`.
• No sabes qué opciones admite un comando: consulta `breeze <comando> --help`.

Cuando ya domines lo básico

Después de crear, compilar, ejecutar y modificar un proyecto, puedes explorar módulos adicionales, dependencias externas, pruebas, documentación y presets de otras plataformas. El catálogo de comandos que sigue está pensado como referencia: no necesitas memorizarlo ni leerlo entero antes de empezar.

PARTE II · CATÁLOGO

Compilar, ejecutar e instalar

Preparación y compilación

Comando	Parámetros y efecto
breeze setup [--required | --all]	Revisa herramientas de compilación y ofrece instalar las que falten. --required instala las obligatorias sin preguntas; --all incluye las opcionales sin preguntas.
breeze build [opciones]	Configura y compila el proyecto a través de los scripts del SDK.
breeze install [opciones]	Configura, compila e instala. --no-build omite la compilación; úsalo solo si los artefactos ya están actualizados.
breeze clean [--no-build] [--docs]	Limpia artefactos. --no-build conserva el árbol de compilación; --docs incluye los archivos de documentación generados.
breeze docs	Genera documentación Doxygen para los módulos del proyecto.

Opciones de build e install

Opción	Descripción
-P, --platform PLATFORM	Plataforma deseada: Linux, macOS o Windows.
-A, --arch ARCH	Arquitectura: x64, x86, arm32 o arm64, según el SDK y el equipo.
-C, --compiler COMPILER	Compilador: GCC, Clang, MSVC, clang-cl, MinGW u otro preset disponible.
-M, --mode MODE	Tipo de compilación: Release, Debug, ReleaseDebugSymbols o MinSizeRel.
-j, --jobs N	Número de trabajos paralelos de compilación.
-p, --preset NAME	Preset CMake exacto, por ejemplo linux-x64-gcc.
--cargs ARG...	Argumentos de configuración CMake, por ejemplo -DBUILD_TESTS=ON.
--bargs ARG...	Argumentos de cmake --build; requiere un directorio configurado previamente.
--no-configure	No ejecuta la fase de configuración; requiere configuración previa. Con esta opción, --cargs se ignora.
--no-build (solo install)	Instala sin llamar a la compilación.

Ejemplo con varias opciones: breeze build -p linux-x64-clang -M Debug -j 8 --cargs -DBUILD_TESTS=ON.

PARTE II · CATÁLOGO

Ejecución y configuración predeterminada

Ejecutar módulos ejecutables

Comando	Descripción
breeze run	Ejecuta el ejecutable predeterminado. Si no hay uno configurado, usa el único ejecutable del proyecto cuando la CLI puede determinarlo; requiere una compilación previa.
breeze run <id>	Ejecuta el módulo ejecutable identificado por su ID; también puede ejecutar pruebas o ejemplos compilados. Primero ejecuta breeze build si aún no existe una compilación.

Ambas formas aceptan -p/--preset, -P/--platform, -A/--arch, -C/--compiler y -M/--mode. El modo no cambia de forma independiente el directorio por preset. Para pasar parámetros al programa, usa -x seguido de los argumentos:

```
breeze run game_exe -x --config config/game.json --verbose
```

Establecer valores predeterminados por proyecto

Comando	Descripción
breeze defaults build platform [--set VALUE]	Consulta o fija la plataforma por defecto.
breeze defaults build arch [--set VALUE]	Consulta o fija la arquitectura por defecto.
breeze defaults build compiler [--set VALUE]	Consulta o fija el compilador por defecto.
breeze defaults build preset [--set VALUE]	Consulta o fija el preset completo por defecto.
breeze defaults build mode [--set VALUE]	Consulta o fija el tipo de compilación por defecto.
breeze defaults run [--set ID]	Consulta o fija el ejecutable que breeze run utiliza si no se proporciona un ID.

Los valores predeterminados se guardan en .breeze/defaults.json; es recomendable versionar ese archivo si quieres que el equipo comparta las mismas preferencias. --set none borra un valor predeterminado.

Ejemplos

```
breeze defaults build preset --set linux-x64-gcc
breeze defaults build mode --set Debug
breeze defaults run --set game_exe
breeze build
```

PARTE II · CATÁLOGO

Administrar módulos y dependencias

Crear módulos

Comando	Opciones principales
breeze add exec <name> [--id ID] [--deps ...]	Crea un módulo ejecutable. El ID predeterminado termina en _exe.
breeze add lib <name> (--static | --shared) [--id ID] [--deps ...]	Crea una biblioteca estática o compartida. Debes elegir exactamente uno de --static y --shared.
breeze add obj <name> [--id ID] [--deps ...]	Crea un módulo de objetos que otro módulo incorpora mediante DEPENDS.
breeze add test <name> [--id ID] [--deps ...]	Crea un ejecutable de prueba; se construye cuando BUILD_TESTS=ON.
breeze add example <name> [--id ID] [--deps ...]	Crea un ejecutable de ejemplo; se construye cuando BUILD_EXAMPLES=ON.

Los IDs automáticos suelen ser el nombre en snake_case con sufijo _exe, _lib, _obj, _test o _example. En --deps puedes indicar IDs de módulos o destinos CMake externos, como fmt::fmt.

Consultar estructura y dependencias

Comando	Descripción
breeze ls [--filter TEXT]	Lista todos los módulos; el filtro busca texto en el ID o en el nombre de la carpeta.
breeze ls [--exec] [--obj] [--lib] [--test] [--example]	Filtra por tipo de módulo. Las opciones de tipo pueden combinarse.
breeze ls --shared / breeze ls --static	Muestra solo bibliotecas compartidas o estáticas; elige uno de los dos filtros.
breeze deps of <module> [--recursive]	Lista los módulos/destinos de los que depende un módulo; con --recursive, sigue la cadena de dependencias.
breeze deps on <module> [--recursive]	Lista los módulos que dependen del indicado; con --recursive, muestra dependientes indirectos.

Cambiar o eliminar módulos

Comando	Descripción
breeze rename <id> [--name NAME] [--id NEW_ID]	Renombra carpeta, ID o ambos y actualiza los DEPENDS que referencian el ID.
breeze delete <id> [--yes]	Elimina el módulo y su carpeta. Se niega si otros módulos dependen de él; --yes omite la confirmación interactiva.

PARTE II · CATÁLOGO

CLI, SDK y configuración global

Administrar las versiones del CLI

Comando	Descripción
breeze cli version	Muestra la versión del CLI que se está ejecutando.
breeze cli ls	Lista las versiones del CLI instaladas localmente.
breeze cli install <version>	Instala una versión del CLI. La versión propia puede instalarse desde el ejecutable actual; otras versiones se obtienen de la fuente configurada.
breeze cli pick <version>	Selecciona qué versión se ejecutará por defecto en llamadas futuras.
breeze cli uninstall <version>	Desinstala una versión instalada.

El CLI se administra globalmente; sus versiones se guardan en ~/.breeze/cli/<version>/ (o bajo $BREEZE_HOME). La versión seleccionada está indicada por cli/current. La carpeta ~/.breeze/bin debe estar en PATH para invocar breeze sin ruta completa.

Administrar las versiones del SDK del proyecto

Comando	Descripción
breeze sdk version	Muestra la versión del SDK usado por el proyecto actual.
breeze sdk update	Consulta versiones disponibles e indica si hay una actualización compatible.
breeze sdk upgrade	Actualiza al SDK más reciente de la misma versión mayor.
breeze sdk migrate <version>	Migra a la versión especificada, incluso entre versiones mayores o desde una estructura antigua.

Configurar fuentes y editor

Comando	Descripción
breeze config editor [--set VALUE]	Muestra o define el editor que usan las herramientas para abrir archivos; --set none borra el valor.
breeze config sdk-source [--set VALUE]	Muestra o define el repositorio Git o directorio de versiones del SDK.
breeze config cli-source [--set VALUE]	Muestra o define la URL/directorio donde se obtienen versiones del CLI.

La configuración global se almacena normalmente en ~/.breeze/config.json. Las variables de entorno BREEZE_EDITOR, BREEZE_SDK_SOURCE y BREEZE_CLI_SOURCE tienen prioridad sobre los valores guardados.

PARTE II · CATÁLOGO

Herramientas externas y scripts del SDK

Instalar y administrar herramientas

Comando	Descripción
breeze tool install <repo>	Instala una herramienta desde un repositorio Git o una carpeta que contiene breeze-tool.json.
breeze tool uninstall <id>	Desinstala una herramienta del proyecto.
breeze tool update <id>	Consulta si existe una versión más reciente.
breeze tool upgrade <id>	Actualiza a la versión más reciente de la misma versión mayor.
breeze tool migrate <id> <version>	Cambia la herramienta a una versión específica.
breeze tool ls	Lista las herramientas del proyecto.
breeze tool <id> <argumentos...>	Ejecuta la herramienta instalada y reenvía todo lo que sigue al ID como argumentos de la propia herramienta.

El manifiesto mínimo breeze-tool.json declara id, version, description y entry. entry puede ser una ruta única o un objeto con rutas distintas para POSIX y Windows. Las herramientas se instalan en .breeze/tools/ y sus metadatos se guardan en .breeze/tools.json (ambos pueden versionarse con el proyecto).

```
{
  "id": "assetpack",
  "version": "1.0.0",
  "description": "Empaqueta recursos",
  "entry": { "posix": "bin/assetpack.sh", "windows": "bin/assetpack.bat" }
}
```

Scripts del SDK (interfaz de bajo nivel)

Los mismos scripts tienen extensión .sh en Linux/macOS y .bat en Windows. Puedes invocarlos directamente desde el proyecto; aceptan -h o --help. Cuando se proporciona --preset, colócalo al principio de los argumentos. El script run requiere una compilación previa; la CLI principal los utiliza para conservar el entorno de compilación apropiado.

Script	Uso principal
setup.sh / setup.bat	[--yes] [--all] [--everything] [--check] [--skip-tools]. Comprueba herramientas, ofrece instalarlas, descarga herramientas fijadas y activa el hook de pre-commit.
build.sh / build.bat	[--preset NAME] [BUILD_TYPE] [CMAKE_ARGS...]; --list-presets lista los presets. BUILD_TYPE: Debug, Release, RelWithDebInfo o MinSizeRel.
run.sh / run.bat	[--preset NAME] TARGET [-- ARGUMENTOS...]. Ejecuta un destino de CMake por nombre; requiere que una compilación previa haya generado el manifiesto y el ejecutable.
install.sh / install.bat	[--preset NAME] [BUILD_TYPE]. Instala los destinos del proyecto ya compilados.
docs.sh / docs.bat	[--preset NAME] [MODULE]. Genera documentación de todos los módulos o de uno.
refactor.sh / refactor.bat	[PATH]. Ejecuta clang-format y luego clang-tidy con --fix; necesita las herramientas y, para clang-tidy, una compilación que haya generado compile_commands.json.

PARTE II · CATÁLOGO

Notas de referencia y ejemplos útiles

Ejemplos de extremo a extremo

Objetivo	Comandos
Proyecto nuevo desde un SDK ya configurado	breeze project new Aurora --sdk 1.0.0 → cd Aurora → breeze setup → breeze build → breeze run
Biblioteca reutilizable y ejecutable	breeze add lib Math --static → breeze add exec Game --deps math_lib → breeze build → breeze run game_exe
Habilitar y ejecutar pruebas	breeze add test MathChecks --deps math_lib → breeze build --cargs -DBUILD_TESTS=ON → ctest --test-dir out/build/<preset> --output-on-failure
Cambiar de compilador	breeze build -P linux -A x64 -C clang o especificar el preset exacto con -p.
Actualizar SDK con control de cambios	git status → breeze sdk update → breeze sdk upgrade → git diff
Generar documentos e instalar	breeze docs → breeze install

Compatibilidad y límites a tener en cuenta

- Los presets disponibles dependen del sistema operativo y de las herramientas instaladas. Consulta cmake --list-presets o breeze/scripts/build.sh --list-presets si una combinación no está disponible (en Windows, usa build.bat).
- Las compilaciones cruzadas pueden necesitar compiladores y bibliotecas del destino; ejecutar esos binarios puede requerir qemu-user o wine.
- En Windows, MSVC y clang-cl necesitan el entorno y componentes de Visual Studio adecuados. Los scripts del SDK preparan el entorno para la ejecución habitual del CLI.
- La actualización del SDK reemplaza breeze/; los cambios a mano dentro de esa carpeta pueden perderse. Pon la personalización del proyecto en project/, source/ y vendor/ en lugar de editar el SDK.
- El script refactor cambia archivos automáticamente. Ejecuta git diff y pruebas antes de aceptar los cambios.

Glosario breve

Término	Definición
CLI	Aplicación de línea de comandos; se invoca como breeze.
SDK	Versión de la plantilla y el sistema de construcción alojados en breeze/.
Preset	Configuración CMake que identifica destino, arquitectura y compilador.
Módulo	Directorio con CMakeLists.txt que declara un ejecutable, biblioteca, objetos, pruebas o ejemplo.
ID de destino	Nombre interno empleado por CMake y por comandos como breeze run o breeze add --deps.
Dependencia	Biblioteca, módulo de objetos o destino externo que se incorpora mediante DEPENDS.

> **Último recurso**
> Ante una diferencia entre esta guía y una versión del programa, ejecuta breeze --help y la ayuda específica del comando. Esta documentación se basa en las versiones incluidas en los archivos fuente proporcionados (SDK 1.0.0).