# Herramientas necesarias

Crear un sistema operativo quiere decir que al menos voy a tener que programar tanto en C como en ensamblador y para esto es necesario tener una serie de herramientas en cuenta

## Tools

- **Assembler**: GNU assembler version 2.44 (i686-elf) using BFD version (GNU Binutils) 2.44
- **Compiler**: gcc version 14.2.0 (GCC)
- **Linker**: GNU ld (GNU Binutils) 2.44
- **Debugger**: GNU gdb (GDB) 16.2
- **Builder**: GNU Make 4.4.1

## Target de compilación

El target de compilación será i686-elf, este es un formato genérico que servirá para definir un target de compilación base. Si desglosamos su nombre este es "Executable linkable format x86 6th gen". Este target de compilación es para 32 bits y es más moderno que el anterior i386 pues incluye algunas instrucciones más.

## Compilando herramientas entorno a i386-elf

Bintuils es un paquete que incluye todas las herramientas relacionadas con el proceso de linkado y ensamblaje. GDB y GCC son herramientas mucho más complejas y cada una tiene su repositorio.

## Compilando Bintuils

Luego de descargar la última versión de Binutils [https://ftp.gnu.org/gnu/binutils/] Hay que compilarlas en este paso se pueden especificar varios parámetros para hacer que las herramientas funcionen para el target que requiero.

Ya que voy a utilizar estos datos para múltiples procesos lo mejor es definir variables para poder reutilizar sus valores:

```
export PREFIX="$HOME/.local/cross"
export TARGET="i686-elf"
export PATH="$PREFIX/bin:$PATH"
```

### Compilación

```
cd binutils-2.44
./configure --target=$TARGET --prefix="$PREFIX" --disable-nls --disable-werror
```

#### Opciones Importantes

1. **–target=i386-elf**: especicación del target de compilación de las herramientas de desarrollo
2. **–prefix="$HOME/.local/cross"**: directorio de instalación de estas herramientas
3. **–disable-werror**: No tratar los "warnings" como errores en tiempo de compilación

#### Opciones Opcionales

1. **–disable-nls**: Reduce el tiempo de compilación deshabilitando soporte para otros idiomas que no sean inglés

### Instalación

```
make install
```

## Compilando GDB (GNU Debugger)

### Compilación

```
cd gdb-16.2
./configure --target=$TARGET --prefix="$PREFIX" --disable-werror
make all-gdb
```

**Opciones**: En este caso no hay opciones que no haya especificado antes.

--disable-nls: no es necesario en este caso

### Instalación

```
make install-gdb
```

# Compilando GCC (GNU cross-compiler)

GCC es un cross-compiler lo que significa que permite generar código compilado desde un equipo A para un equipo B (target).

## Compilación

**Compilar gcc:**

```
mkdir build-gcc
pushd buid-gcc
../configure --target=$TARGET --prefix="$PREFIX" --disable-nls --enable-languages=c,c++
make all-gcc
```

**Opciones**

1. **–enable-languages**: Lenguajes a habilitar en este caso he habilitado también c++ pero no estoy seguro de que lo vaya a usar, quizás para abstracciones de alto nivel...

**Opciones implicitas** Estas opciones eran necesarias para versiones antiguas de gcc en caso de querer que gcc funcione en un entorno freestanding. Estoy compilando GCC en su última versión y este actualmente infiere que es un entorno freestanding por el target de compilación y el tipo de ensamblador. Igualmente considero importante listarlas porque definen muy bien como va a ser el cross compiler resultante:

1. **–without-headers**: GCC no dependerá los headers de la librería estandar por defecto esto es importante porque no podré usar gran parte de la librería estandar, solo podré usar la librería estándar dentro de la parte freestanding, más sobre eso en breve.
2. **–disable-hosted-libstdcxx**: Compilar GCC sin soporte para la parte de la librería estándar clasificada como hosted, más sobre eso en breve.

### Compilar librería standar c: (freestanding)

```
make all-target-libgcc
```

### Compilar librería standar c++: (freestanding)

Es posible que no use c++ en ningún momento pero prefiero tener la capacidad de usarlo, sin requerir volver a pasar por el proceso de configuración del cross compiler:

```
make all-target-libstd++-v3
```

## Instalación

```
make install-gcc
make install-target-libgcc
make install-target-libstdc++-v3
```

# Entorno Hosted vs Freestanding

El estándar de c define dos entornos de ejecución diferentes dependiendo de donde se vaya a ejecutar el binario resultante de la compilación.

## Hosted:

Es un entorno donde se presupone que existen todas las dependencias de la librería estandar, en otras palabras nuestro programa se ejecuta en el espacio de usuario de un sistema operativo y cuenta con todas las abstracciones proporcionadas por el kernel para realizar tareas.

## Freestanding:

En este caso no se cuenta con ninguna librería y no existe nada que nos proporcione abstracciones.

**¿Si es así como si quiera existen si no se puede incluir estas librerías?** Estas librerías no existen en el sistema, las genera el compilador en tiempo de compilación y incluyen macros y definiciones de tipos básicas.

**stdlid (freestanding)**:

- float.h
- iso646.h
- limits.h
- stdaling.h
- stdarg.h
- stdbool.h
- stdef.h

- stdint.h
- stdnoreturn.h

Todas estas librerías son definiciones o macros, que se pueden resolver si un solo object file en otras palabras no requieren implementación solo definición.

**Ejemplo**: En el caso de stdint.h podría ser más o menos esto:

```
typedef signed char int8_t;
typedef unsigned char uint8_t;
typedef short int16_t;
typedef unsigned short uint16_t;
typedef int int32_t;
typedef unsigned int uint32_t;
typedef long long int64_t;
typedef unsigned long long uint64_t;


typedef long long intmax_t;
typedef unsigned long long uintmax_t;
```

**Ejemplo**: En el caso de stdbool.h

```
#define bool _Bool // _Bool es una keyword desde C99 que solo permite 0 o 1, pero el tamaño total de un bool sigue siendo 1 Byte por
#define true 1
#define false 0
```

## Configuración final

Para poder trabajar cómodamente en el proyecto añadiré a mi .zshrc (uso zsh como shell principal) el PATH modificado para incluir mis nuevas herramientas para desarrollar el sistema operativo

```
export PATH=$PATH:$HOME/bins:/usr/local/bin:$HOME/go/bin:$HOME/.local/share/gem/ruby/3.0.0/bin:$HOME/.local/bin:$HOME/.cargo/bin:/ho
```