Emscripten SDK (emsdk)

Emscripten SDK (`emsdk`) is used to perform all SDK maintenance. You only need to install the SDK once; after that emsdk can do all further updates!

With *emsdk* you can download, install or remove *any* SDK or Tool, and even use the bleeding edge versions in development on GitHub. To access the *emsdk* on Windows, first launch the Emscripten Command Prompt. Most operations are of the form `./emsdk command`.

This document provides the command syntax, and a set of guides explaining how to perform both common and advanced maintenance operations.

Command line syntax

./emsdk [**help** | **list** [**--old**] | **update** | **install** <*tool/sdk*> | **uninstall** <*tool/sdk*> | **activate** <*tool/sdk*>]

Arguments

Command	Description
`list [--old]`	Lists all current SDKs and tools and their installation status. With the `--old` parameter, historical versions are also shown.
`update`	Fetches the latest list of all available tools and SDKs (but does not install them).
`install <tool/sdk>`	Downloads and installs the specified tool or SDK.
`uninstall <tool/sdk>`	Removes the specified tool or SDK from the disk.
`activate <tool/sdk>`	Sets the specified tool or SDK as the default tool in the system environment. On Linux you additionally have to enable the environment settings using `source ./emsdk_env.sh`, as described in "How do I change the currently active SDK version?".
`help`	Lists all supported commands. The same list is output if no command is specified.

Note

For Linux and macOS the commands are called with **./emsdk**. On Windows use **emsdk**.

Tools and SDK targets

The `<tool/sdk>` given above as a command argument is one of the targets listed using `./emsdk list` (or `./emsdk list --old`).

Note that some of the tools and SDK names include *master* or *main*: these targets are used to clone and pull the very latest versions from the Emscripten main and master branches.

You can also specify a target of `latest` to grab the most current SDK.

SDK concepts

The Emscripten toolchain includes a number of different tools, including *Clang*, *Emscripten*, *Java*, *Git*, *Node*, etc. *Emsdk* is a package manager for controlling which tools are installed, and from the set of installed tools, which are active.

The current set of available tools and SDKs are listed using `./emsdk list`. These can be installed individually (`./emsdk install node-9.10.17-64bit`) or as a group (`./emsdk install node-9.10.17-64bit java-7.45-64bit`).

The SDK targets are a convenience mechanism for specifying the full set of tools used by a particular Emscripten release. For example, the two lines below are equivalent:

```
./emsdk install sdk-upstream-main-64bit
./emsdk install git-2.8.1 clang-upstream-main-64bit node-9.10.17-64bit python-2.7.5.3-64bit java-7.45-64bit llvm-g
```

A particular installed SDK (or tool) can then be set as active, meaning that it will be used when Emscripten is run. The active "compiler configuration" is stored is a config file (*.emscripten*) within the emsdk directory.

Note

The different tools and SDKs managed by *emsdk* are stored in different directories under the root folder you specified when you first installed an SDK, grouped by tool and version.

Emscripten Compiler Configuration File (.emscripten)

The *Compiler Configuration File* stores the active configuration on behalf of the *emsdk*. The active configuration defines the specific set of tools that are used by default if Emscripten in called on the Emscripten Command Prompt.

The configuration file is named **.emscripten**. It is *emsdk*-specific, so it won't conflict with any config file the user might have elsewhere on their system.

The file should generally not be updated directly unless you're building Emscripten from source. Instead, use the *emsdk* to activate specific SDKs and tools as needed (`emsdk activate <tool/SDK>`).

Below are examples of possible **.emscripten** files created by *emsdk*. Note the variable names used to point to the different tools:

```
# .emscripten file from Windows SDK

import os
LLVM_ROOT='C:\Program Files\Emscripten\clang\e1.37.9.64bit'
NODE_JS='C:\Program Files\Emscripten/node\8.10.17_64bit\node.exe'
```

```
# .emscripten file from Linux SDK

import os
NODE_JS = 'node/8/'
LLVM_ROOT='/home/ubuntu/emsdk/upstream/bin'
```

"How to" guides

The following topics explain how to perform both common and advanced maintenance operations, ranging from installing the latest SDK through to installing your own fork from GitHub.

Note

The examples below show the commands for Linux and macOS. The commands are the same on Windows, but you need to replace **./emsdk** with **emsdk**.

How do I just get the latest SDK?

Use the `update` argument to fetch the current registry of available tools, and then specify the `latest` install target to get the most recent SDK:

```
# Fetch the latest registry of available tools.
./emsdk update

# Download and install the latest SDK tools.
./emsdk install latest

# Set up the compiler configuration to point to the "latest" SDK.
./emsdk activate latest
```

How do I use emsdk?

Use `./emsdk help` or just `./emsdk` to get information about all available commands.

How do I check which versions of the SDK and tools are installed?

To get a list of all currently installed tools and SDK versions (and all available tools) run:

```
./emsdk list
```

A line will be printed for each tool and SDK that is available for installation. The text `INSTALLED` will be shown for each tool that has already been installed. If a tool/SDK is currently active, a star (*) will be shown next to it.

How do I install a tool/SDK version?

Use the `install` argument to download and install a new tool or SDK version:

```
./emsdk install <tool/sdk name>
```

For example:

```
./emsdk install sdk-1.38.27-64bit
```

Note

An installed tool is present on the local machine, but not necessarily the active environment. To make an installed SDK active, use the `activate` command.

How do I remove a tool or an SDK?

Use the `uninstall` argument to delete a given tool or SDK from the local computer:

```
./emsdk uninstall <tool/sdk name>
```

If you want to completely remove Emscripten from your system, follow the guide at Uninstalling the Emscripten SDK.

How do I check for updates to the Emscripten SDK?

First use the `update` command to fetch package information for all new tools and SDK versions. Then use `install <tool/sdk name>` to install a new version:

```
# Fetch the latest registry of available tools.
./emsdk update

# Download and install the specified new version.
./emsdk install <tool/sdk name>
```

How do I change the currently active SDK version?

Toggle between different tools and SDK versions using the `activate` command. This will set up `.emscripten` to point to that particular tool:

```
./emsdk activate <tool/sdk name>

# On Linux and macOS, also set the environment variables.
source ./emsdk_env.sh
```

Note

On Linux and macOS, `activate` writes the required information to the configuration file, but cannot automatically set up the environment variables in the current terminal. To do this you need to call `source ./emsdk_env.sh` after calling `activate`. The use of `source` is a security feature of Unix shells.

On Windows, calling `activate` automatically sets up the required paths and environment variables.

Note

If you add `./emsdk_env.sh` to you default shell config *emsdk* tools (including the emsdk version of node) will be added to your PATH and this could effect the default version of node used on your system.

How do I install and activate old Emscripten SDKs and tools?

Emsdk contains a history of old tools and SDKs that you can use to maintain your migration path. Use the `list --old` argument to get a list of archived tool and SDK versions, and `install <name_of_tool>` to install a specific tool:

```
# Get list of the old versions of the tool.
./emsdk list --old

# Install the required version.
./emsdk install <name_of_tool>

# Activate required version.
./emsdk activate <name_of_tool>
```

How do I track the latest changes with the SDK?

To try the latest changes with emsdk you can install and activate a special version called `tot` (Tip-Of-Tree) which is continuously built and usually contains Emscripten and LLVM changes just a few hours after they are committed:

```
./emsdk install tot
./emsdk activate tot
```

If you want to build everything yourself from the very latest sources you can use `sdk-main-64bit`:

```
# Install git (skip if the system already has it).
./emsdk install git-2.8.1

# Clone+pull the latest emscripten-core/emscripten/main.
./emsdk install sdk-main-64bit

# Set this as the active version.
./emsdk activate sdk-main-64bit
```

How do I use my own Emscripten fork with the SDK?

It is also possible to use your own fork of the Emscripten repository via the SDK. This is useful in the case when you want to make your own modifications to the Emscripten toolchain, but still keep using the SDK environment and tools.

To to this all you need to do is set the `EM_CONFIG` environment variable to point to the emsdk emscripten config and then put your own checkout of emscripten first in the `PATH`:

```
cd my_emscripten/

# Tell emscripten to use the emsdk config file
export EM_CONFIG=/path/to/emsdk/.emscripten

# Now your version of emscripten will use LLVM and binaryen
# binaries from the currently active version of emsdk.
./emcc
```