

Development Environment

Core Emscripten is a python script whose repository can be cloned from GitHub sources.

```
root@linux:~# git clone https://github.com/emscripten-core/emsdk.git
Cloning into 'emsdk'...
/usr/lib/git-core/git-remote-https: /usr/local/lib/libldap_r-2.4.so.2: no version information available (required by /lib/x86_64-linux-gnu/libcurl-gnutls.so.4)
/usr/lib/git-core/git-remote-https: /usr/local/lib/liblber-2.4.so.2: no version information available (required by /lib/x86_64-linux-gnu/libcurl-gnutls.so.4)
remote: Enumerating objects: 2884, done.
remote: Counting objects: 100% (146/146), done.
remote: Compressing objects: 100% (87/87), done.
remote: Total 2884 (delta 89), reused 95 (delta 59), pack-reused 2738
Receiving objects: 100% (2884/2884), 1.54 MiB | 243.00 KiB/s, done.
Resolving deltas: 100% (1841/1841), done.
root@linux:~# _
```

Figure 1:GitHub clone.

The latest SDK tools can be install using 'install latest' command in the cloned repository.

```
root@linux:~/emsdk# ./emsdk install latest
Resolving SDK alias 'latest' to '2.0.30'
Resolving SDK version '2.0.30' to 'sdk-releases-upstream-c69458f1bbf3ef5b8da4e934de210659cc9bca04-64bit'
Installing SDK 'sdk-releases-upstream-c69458f1bbf3ef5b8da4e934de210659cc9bca04-64bit'..
Installing tool 'node-14.15.5-64bit'..
Downloading: /root/emsdk/zips/node-v14.15.5-linux-x64.tar.xz from https://storage.googleapis.com/webassembly/emscripten-releases-builds/deps/node-v14.15.5-linux-x64.tar.xz, 21391232 Bytes
-
```

Figure 2:Install SDK tools.

Latest installed SDK can be made available to current user by activating. Instructions are written to '.emscripten' file located at the current user's directory.

```
root@linux:~/emsdk# ./emsdk activate latest
Resolving SDK alias 'latest' to '2.0.30'
Resolving SDK version '2.0.30' to 'sdk-releases-upstream-c69458f1bbf3ef5b8da4e934de210659cc9bca04-64bit'
Setting the following tools as active:
  node-14.15.5-64bit
  releases-upstream-c69458f1bbf3ef5b8da4e934de210659cc9bca04-64bit

Next steps:
- To conveniently access emsdk tools from the command line,
  consider adding the following directories to your PATH:
  /root/emsdk
  /root/emsdk/node/14.15.5_64bit/bin
  /root/emsdk/upstream/emscripten
- This can be done for the current shell by running:
  source "/root/emsdk/emsdk_env.sh"
- Configure emsdk in your shell startup scripts by running:
  echo 'source "/root/emsdk/emsdk_env.sh"' >> $HOME/.bash_profile
root@linux:~/emsdk#
```

Figure 3:Activating latest SDK.

Active PATH and environmental variables for the current terminal available in the user context are set using source command.

```
root@linux:~/emsdk# source ./emsdk_env.sh
Adding directories to PATH:
PATH += /root/emsdk
PATH += /root/emsdk/upstream/emscripten
PATH += /root/emsdk/node/14.15.5_64bit/bin

Setting environment variables:
PATH = /root/emsdk:/root/emsdk/upstream/emscripten:/root/emsdk/node/14.15.5_64bit/bin:/home/sp1d3r/em
dk/upstream/emscripten:/home/sp1d3r/emsdk/node/14.15.5_64bit/bin:/home/sp1d3r/emsdk:/root/jdk1.8.0_101
/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin:/usr/lib/postgresql/10/bin
EMSDK = /root/emsdk
EM_CONFIG = /root/emsdk/.emscripten
EMSDK_NODE = /root/emsdk/node/14.15.5_64bit/bin/node
root@linux:~/emsdk#
```

Figure 4:Environmental variables.



```
root@linux:~/emsdk# emcc -v
emcc (Emscripten gcc/clang-like replacement + linker emulating GNU ld) 2.0.30 (f782b50a7f8dded7cd0e2c7
ee4fed41ab743f5c0)
clang version 14.0.0 (https://github.com/llvm/llvm-project c4048d8f50aaf2c4c13b8d3e138abc34a22da754)
Target: wasm32-unknown-emscripten
Thread model: posix
InstalledDir: /root/emsdk/upstream/bin
root@linux:~/emsdk# _
```

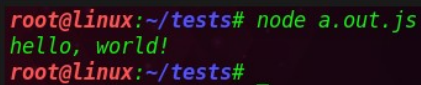
Figure 5:Successfully install emcc.



```
GNU nano 3.2      hello_world.c      Modified
#include <stdio.h>

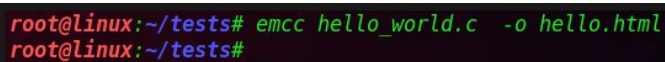
int main() {
    printf("hello, world!\n");
    return 0;
}
_
```

Figure 6:Hello world project.



```
root@linux:~/tests# node a.out.js
hello, world!
root@linux:~/tests# _
```

Figure 7:Generated JavaScript output.



```
root@linux:~/tests# emcc hello_world.c -o hello.html
root@linux:~/tests# _
```

Figure 8:Generating HTML output.

Transpiling code base

A shell sort algorithm developed in c++ is transpiled in this case to native code.

```
root@linux:~/tests# emcc shell_sort.cpp -s EXIT_RUNTIME=1
root@linux:~/tests# _
```

Figure 9:Building code base.

```
root@linux:~/tests# node a.out.js
Array before sorting:
12 34 54 2 3
Array after sorting:
2 3 12 34 54
root@linux:~/tests#
```

Figure 10:Generated JavaScript.

```
root@linux:~/tests# emcc -O2 shell_sort.cpp -o shell.html -s EXIT_RUNTIME=1
root@linux:~/tests# _
```

Figure 11:Generating optimized HTML.

A simple python or PHP web server can be used to execute generated HTML code. Results can be viewed using a web browser.

```
Array before sorting:  
12 34 54 2 3  
Array after sorting:  
2 3 12 34 54
```

Figure 12:Using a web browser.

Performance Analysis

Code compiled on Web-Assembly runs at 1.55 times slower than native code. This is due to the following reasons:

- Code contained in Web-Assembly contains 2 times more loads and stores than native code.
- Web-Assembly requires more safety dynamic checks and therefore contains more branches than native code.

Appendix A

```
#include <iostream>
using namespace std;

/* shellSort */
int shellSort(int arr[], int n)
{
    for (int gap = n/2; gap > 0; gap /= 2)
    {
        for (int i = gap; i < n; i += 1)
        {
            int temp = arr[i];

            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];

            arr[j] = temp;
        }
    }
    return 0;
}
```



```
void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

int main()
{
    int arr[] = {12, 34, 54, 2, 3}, i;
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Array before sorting: \n";
    printArray(arr, n);

    shellSort(arr, n);

    cout << "\nArray after sorting: \n";
    printArray(arr, n);

    return 0;
}
```