This manual was extracted directly from http://www.ohwr.org/projects/hdl-make/wiki/Quick_start, the original project on which this one was based.

Quick start

Hdlmake aims to be a multi-purpose tool for HDL development. It can put together projects made of several modules, create project files for ISE or Quartus, run synthesis flows, run simulations in Modelsim, run synthesis on remote machines and more. All these things can be done in two ways: by running hdlmake or by generating a makefile containing all needed commands. It's up to you which way you choose.

Before you can do all these fancy things, you have to describe your project in a special way. We tried to avoid making too much hassle.

First of all, think about your project. The most basic thing is to think about your project: is it a monolithic creation? Is it composed of many small modules that you would like too put in different repositories, but for some reasons you didn't? Maybe you would like to divide your design into smaller ones and maintain them separately?

If your answer was positive, go to paragraph How to divide a design and apply hdlmake.
If your project is simple enough our you are just prefer to keep everything in one place, read following paragraph (How to apply hdlmake)

Before you start

If you want to synthesize project for ISE, please be sure that the environmental variable XLINX is set in your environment. If you have a correct ISE installation it should be permanently set, but for some reasons may be not. XILINX should indicate yo ur ISE catalogue. Its value is version-dependend, but for most of the versions it looks similiar. With i.e. ISE v12.1 it should be set to/opt/Xilinx/12.1/ISE_DS

How to apply hdlmake

The only thing you have to do is to create Manifest's. These are Python files that describe your project's structure and some options. The structure is the same, whatever you want to do, but options varies, depending on what you want to do (simulate/synthesis). Let's do it for synthesis.

How to apply hdlmake for synthesis

Let's say that you want to apply hdlmake to a single module (kept in a single repository). Go to the place where your top HDL file is. Touch a file called Manifest.py and edit it.
What you need to do, is to list there all files that are needed in synthesis. That includes sources (.vhd, .v, .sv), constraints files (.ucf) and whatever you consider needed.
Define a python list that contains paths to all files that you need, for example:

```
files = ["../hdl/file1.vhd", "../hdl/file2.vhd", "../hdl1/file.v"]
```

If you want to put this files into a library (in the VHDL's sense), you have to use *library* variable:

```
library = "mylib"
```

When Hdlmake detects a library, it automatically puts it into the makefile. Corresponding library is made and files from the library are mapped into it.

Typing this "../hdl" prefix can be annoying. If you want to avoid it, you can do one of two things:

1. Treat hdl and hdl1 as separate modules. Then you need to list them as modules:

```
modules = {"local" : ["../hdl", "../hdl1"]}
```

Each module needs to have its Manifest, so you must create one for each module.
The syntax is a bit complicated. What you see here is a python dictionary with "local" key. Its value is a python list, with paths to modules listed. This dictionary can have in general three keys: "local", "git" and "svn". This values indicate where particular modules are stored.

2. In files list folders instead of files. This is a bit dangerous, since when Hdlmake encounters a folder, it takes all files that it contains. If you by accident put a wrong file in the directory, it will show up in your project files. Be careful.

Once you are ready with the files, you have to add other variables. Your Manifest can look then like this:

```
target = "xilinx"
action = "synthesis"

syn_device = "xc6slx150t"
syn_grade = "-2"
syn_package = "fgg676"
syn_top = "vme64xcore_top_reg"
syn_project = "test.xise"

files = ["top.vhd", "../hdl/file1.vhd", "../hdl/file2.vhd", "../hdl/sv/file.sv"]
```

- target indicates target architecture of a design. It can be equal to either "xilinx" or "altera"
- action indicates the purpose of a manifest. It can be equal to either "simulation" or "synthesis"
- syn_device is number of used part
- syn_grade is speed grade of target FPGA
- syn_package is package variant of target FPGA
- syn_top is project's top entity
- syn_project is the name of IDE's project file that should be created

If you need a hint about Manifest variables, run hdlmake --manifest-help.
If your Manifest looks like above (modules, files, or both listed, synthesis variables), you are ready to go.
There are plenty of things that Hdlmake can do with this project. If you don't want to specify any particular, just run hdlmake with no arguments.
In this case hdlmake does three things:

- downloads all unfetched modules (from svn and git)
- creates project files
- generates multi-purpose makefile

In the makefile you can find three major targets:

- local runs local synthesis
- remote runs remote synthesis. In order to run it, you have to set environmental variables: HDLMAKE_USER andHDLMAKE_SERVER (use shell's export command). You can also hard code in the makefile. To do so, use --synth-server and --synth-user arguments, when running Hdlmake.
- fetch clones repositories or (if they are present) pulls the newest version from the repo.

If you need to add some extra option to the synthesis (like optimization grade, generating ascii file or anything else supported by ISE), open your project in ISE, tick desired option and apply. This will be saved in .xise file. Hdlmake, whenever you run it, will leave this option untouched. And this will be applied in the next synthesis. There is no need to redo makefiles.

How to divide a design and apply Hdlmake

In order to make your projects fit into modularity proposed by hdlmake, think about the modules you would like to distinguish.
Now, put all modules in separate directories. In each of these directories create a manifest that describes its files and necessary modules (it must be located right in the main module's folder; hdlmake doesn't do a deeep scan).

You can start with doing these things locally (it means that in Manifests you have to give local paths). Having your project tested, you can put the modules into repositories.

How to apply Hdlmake simulation

This thing is very similar to using Hdlmake for synthesis. This time, instead of creating a Manifest in the top synthesis folder, you have to create one in the testbench directory. You have to list all modules and files like it was previously. Specify also actionvariable, but this set it to "simulation". Here is an illustrative Manifest:

```
action = "simulation"

modules = { "local" : [ "../../../top/sfpga/pablo_vme",
"../../../modules/vfc_board_wrapper" ] }
files = "main.sv"

vlog_opt= "+incdir+../../../include"
```

In this manifest modules and files are listed, as well as they were listed in synthesis Manifests. *action* is set to "simulation" as mentioned.
A new thing is *vlog_opt*. This variable allows adding custom vlog options, that should be placed in a makefile. You can also use vsim_opt, vcom_opt, vmap_opt.

And if there is a lot of stuff to remember...

...then don't forget that you can always use built-in help:
hdlmake --manifest-help and
hdlmake --help.