Tutorial

Toshio Sekiya

December 8, 2020

Contents

Abstract

This tutorial has nine sections. The main contents is described in the sections from one to eight. Section nine is the copy of Readme.md, which is included in the Buildtools source files.

1 Installation

1.1 Prerequisite

Buildtools requires the following items.

1. Linux OS and bash
2. LaTeX system
3. Make or Rake

1.1.1 Linux OS and bash

Buildtools is tested on Debian and Ubuntu. However, it probably works on other linux distributions. Bash is required because the shell scripts in Buildtools include bash commands.

1.1.2 LaTeX system

There are two options to install LaTeX.

One is installing the LaTeX applications provided by your distribution. If your distribution is Ubuntu, you can install it by typing the following line.

```
$ sudo apt-get install texlive-full
```

If you have another distribution, refer to the distribution's document to install.

The other way is installing TexLive from https://www.tug.org/texlive Refer the documentations in the web to install TexLive system.

1.1.3 Make or rake

These applications are not necessarily required to run the tools in Buildtools. However, it is recommended that they should be used under the control of make or rake. You don't need to install both of them. Choose one which you like.

Make is a traditional build tool originally aimed at C compiler. In Ubuntu, type the following line to install make.

```
$ sudo apt-get install make
```

Rake is a build tool similar to make. It is one of the ruby application. The advantage to use rake is that you can put any ruby codes into Rakefile, which is the script file of rake. Generally speaking, Rakefile is easy to understand than Makefile. In Ubuntu, type the following line to install rake.

```
$ sudo apt-get install rake
```

If you want to install the latest version of ruby, use rbenv and ruby-build. See the following github repository and refer to the documentations there.

- https://github.com/rbenv/rbenv
- https://github.com/rbenv/ruby-build

1.2 Installation

1.2.1 Download

First, access the following github repository.

- https://github.com/ToshioCP/LaTeX-BuildTools

Click the `Code` button, then popup menu appears. Click `DOWNLOAD ZIP` menu. Unzip the download zip file.

1.2.2 Installation

Open your terminal. Change your current directory into the directory you extracted the zip file. Then type:

```
$ bash install.sh
```

This script installs the executable files into `\$HOME/bin`. Debian and Ubuntu adds the directory `\$HOME/bin` into `PATH` environment variable if it exists at the login time. The script makes the directory `\$HOME/bin` if it doesn't exist. In that case, you need to re-login to put the directory into the `PATH` environment variable. This installs scripts into your private directory, so any other users can't access the scripts. This is called user level installation or private installation.

If you want to install the scripts in `/user/local/bin`, you need to have root privilege. If your OS is ubuntu, then type:

```
$ sudo bash install.sh
```

2 Run lb to compile tex files

2.1 First step

Lb is the main script in Buildtools. This section describes how to use it with a small example.

First, make a directory named `example` and change your current directory to it.

```
$ makedir example
$ cd example
```

Then make a tex source file in the directory. Run your favourite editor and copy the following text, then save it as the name `main.tex`.

```
\documentclass{article}
\begin{document}
Hello \LaTeX !!
\end{document}
```

Then, just type `lb`.

```
$ lb
```

Then, it runs latexmk and pdflatex and compile `main.tex` with them. Messages appear on your screen, and that shows the process of the compilation. If there is a line

```
Output written on _build/main.pdf (1 page, 19263 bytes).
```

then the compilation completes correctly. Check the directory.

```
$ ls -l
total 8
drwxrwxr-x 2 user user 4096 Dec  6 11:59 _build
-rw-rw-r-- 1 user user   72 Dec  6 11:59 main.tex
```

A new directory `_build` is generated. Look at the files in the directory.

```
$ cd _build
$ ls
```

There are auxliary files and the target file `main.pdf`. See `main.pdf` with your pdf-viewer, for example evince.

```
$ evince main.pdf
```

Hello LaTeX!!

2.2 Use lb.conf

In the previous subsection, lb runs pdflatex. The reason why lb chose pdflatex is the documentclass 'article'. It can also be compiled by lualatex or xelatex, but pdflatex has been a standard latex engine for ages.

If you want to use, for example, lualatex to compile, you need to specify it in `lb.conf`. This configuration file has six items.

rootfile Rootfile is the main tex file, which usually includes \begin{document} and \end{document}. Other tex files are called 'subfile'.

builddir This is a temporary directory includes all the auxliary files and the target file, which is usually a pdf file.

engine This specifies a latex engine, which is one of pdflatex, xelatex, lualatex, latex and platex.

latex_option This specifies options to give `latexmk`. The option '-halt-on-error' is given to `latexmk` even if lb.conf doesn't exist.

dvipdf This is a program which converts dvi into pdf, which is used only with latex or platex. It is unnecessary with other latex engines. 'dvipdfmx' is the best so far.

preview Pdf viewer. This is used to preview the pdf file when lb is given a subfile as an argument.

Run your editor and type the following and save it as the name `lb.conf`.

```
rootfile=main
builddir=_build
engine=lualatex
latex_option=-halt-on-error
dvipdf=
preview=evince
```

Then, type

```
$ lb
```

Then, it uses lualatex to compile.

If you want to change the name of the tex file to 'example.tex', then modify the first line in lb.conf to

```
rootfile=example
```

or

```
rootfile=example.tex
```

The suffix can be left out.

In addition, if you want to put all the axiliary files and the target file in the source directory, change the second line in lb.conf to:

```
builddir=
```

This specifies null string for builddir item and that means no build directory is made.

Let's try to run `lb` with the following `lb.conf`

```
rootfile=example
builddir=
engine=latex
latex_option=-halt-on-error
dvipdf=dvipdfmx
preview=evince
```

Now, the engine is latex and dvipdf program is dvipdfmx.

```
$ rm -r _build
$ mv main.tex example.tex
$ lb
```

Then messages appear. It includes the following line.

```
This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020)
 (preloaded format=latex)
  ... ...
  ... ...
Output written on example.dvi (1 page, 332 bytes).
Transcript written on example.log.
Latexmk: Examining 'example.log'
=== TeX engine is 'pdfTeX'
Latexmk: Log file says output to 'example.dvi'
Latexmk: All targets (example.dvi) are up-to-date
example.dvi -> example.pdf
[1]
3662 bytes written
```

This tells us that the engine was 'latex'[1]. It generates dvi file instead of pdf file. After that, dvipdfmx is run by latexmk and it translates the dvi file into a pdf file. The name 'dvipdfmx' doesn't appear in the message but 'example.dvi -> example.pdf' is outputed by dvipdfmx. So we know that dvipdfmx was run by latexmk in the build process.

```
$ ls
example.aux  example.fdb_latexmk  example.log  example.tex
example.dvi  example.fls          example.pdf  lb.conf
```

There's no temprary directory like _build because we specified null string for builddir.

One of the important feature of lb is compiling a subfile separately. This will be explained in the section 5 'Test compile' (p. 11).

3 Generate templates

3.1 newtex.conf

The script 'newtex' makes a directory and generates template files in it. This is used at the beginning of the work.

First, a configuration file 'newtex.conf' needs to be made. There is a template file included in the Buildtools source files.

[1] In Texlive2020, 'latex' command calls 'pdftex' instead of 'tex' which is the original TeX program.

```
# This is a configuration file for newtex.
# The name of this file is newtex.conf
# A string between # and new line is a commnet and it is ignored
 by newtex.
# Empty line is also ignored.

title="Tutorial"
# document name

# lb.conf
# Lb.conf has six lines.
# The following six lines are copied to lb.conf.
rootfile=main.tex
builddir=_build
engine=pdflatex
latex_option=-halt-on-error
dvipdf=
preview=evince

# documentclass
documentclass=article

# chapters/sections and subfile names
#   Chapters/sections and subfile names must be surrounded by
 double quotes.
#   Subfile names have no suffix or ".tex" suffix.
# If your LaTeX file is not big and no subfile is necessary, then
 leave out the following lines.
section="Installation" "installation"
section="Run lb to compile tex files" "lb"
section="Generate templates" "generate_templates"   # Subfiles
 are NOT allowed to include space characters. Use underscore
 instead of space.
section="Edit tex files" "edit_tex_files"
section="Test compile" "test_compile"
section="Preprocessing" "preprocessing"
section="Use rake" "rake"
section="Make tarball" "tarball"
```

In this tutorial, I want to show you how to make this tutorial pdf file with Buildtools. The file above is exactly the same as the newtex.conf file as I used to make it.

A string after hash mark (#) in a line is comment and it is ignored by newtex. Empty lines are also ignored. The remaining lines are instructions to newtex. Each line has a 'key=value' structure. The keys are:

title The title of the document you make.

rootfile The name of the rootfile.

builddir The name of the build directory

engine A latex engine to compile source files

latex_option The options you want to give to the latex engine

dvipdf A program that converts dvi to pdf.

preview A pdf viewer

documentclass The name of the documentclass you want to use

chapter Chapters and corresponding subfiles

section Sections and corresponding subfiles

If you make a book (big document) and use book documentclass, use 'chapter' and 'section' key. If you make an article (small document) and use article documentclass, use 'section' key only.

3.2 Run newtex

After you finish editing newtex.conf, just type:

```
$ newtex
```

Then, newtex makes a directory of which the name is 'Tutorial', which is the same as the title in newtex.conf. If the title includes space characters, they are converted to underscore. For example, a title 'A tutorial for beginners' is converted to 'A_tutorial_for_beginners'. This is because a file name includes space character sometimes causes problems. Newtex also generates temlate files under the directory.

```
$ cd Tutorial
$ ls
Makefile            generate_templates.tex  main.tex
Rakefile            helper.tex              preprocessing.tex
cover.tex           installation.tex        rake.tex
edit_tex_files.tex  lb.conf                 tarball.tex
gecko.png           lb.tex                  test_compile.tex
```

Look at some important files.

```
$ cat lb.conf
rootfile=main
builddir=_build
engine=pdflatex
latex_option=-halt-on-error
dvipdf=
preview=evince
```

The contents of this file is the copy of the part of newtex.conf.

```
$ cat main.tex
\documentclass{article}
\input{helper.tex}
\title{Tutorial}
\author{} % Write your name if necessary.
\begin{document}
\maketitle
% If you want a table of contents here, uncomment the following
 line.
%\tableofcontents

\section{Installation}
  \input{installation.tex}
\section{Run lb to compile tex files}
  \input{lb.tex}
\section{Generate templates}
  \input{generate_templates.tex}
\section{Edit tex files}
  \input{edit_tex_files.tex}
\section{Test compile}
  \input{test_compile.tex}
\section{Preprocessing}
  \input{preprocessing.tex}
\section{Use rake}
  \input{rake.tex}
\section{Make tarball}
  \input{tarball.tex}
\end{document}
```

The first line specifies a documentclass which is the same as the value of documentclass key in newtex.conf. The second line has an input command which includes 'helper.tex'. Helper.tex has a role to include packages with `\usepackage` command, define macros with `\newcommand` command and so on. Most of the lines in the preamble are described in helper.tex. It is a good idea to make your own helper.tex because users often use the same preamble in different documents. If you have your helper.tex, copy and overwrite this file.

You need to edit the fourth line. For example,

```
\author{Toshio Sekiya}
```

You can add '\date', '\thanks', '\begin{abstract}' and '\end{abstract}' if you like. If you want to make a table of contents, then uncomment the eighth line. After that, the lines are sections and \input commands to include subfiles.

Rakefile contains instructions to rake. You don't need to modify it so far. Try to use it.

```
$ rake
 ... ...
 ... ...
$ ls
Makefile            gecko.png               main.tex
Rakefile            generate_templates.tex  preprocessing.tex
Tutorial.pdf        helper.tex              rake.tex
_build              installation.tex        tarball.tex
cover.tex           lb.conf                 test_compile.tex
edit_tex_files.tex  lb.tex
$ ls _build
main.aux  main.fdb_latexmk  main.fls  main.log  main.out  main.pdf
$ evince Tutorial.pdf
```

Rake ran lb to compile main.tex and after that it copied _build/main.pdf to Tutorial.pdf. Evince shows Tutorial.pdf as follows.

Tutorial

December 8, 2020

4 Edit tex files

There are eight sections and subfiles. Each subfile is empty just after it is generated by newtex. Editing subfiles is the main work and you need to allocate most of your time to this work.

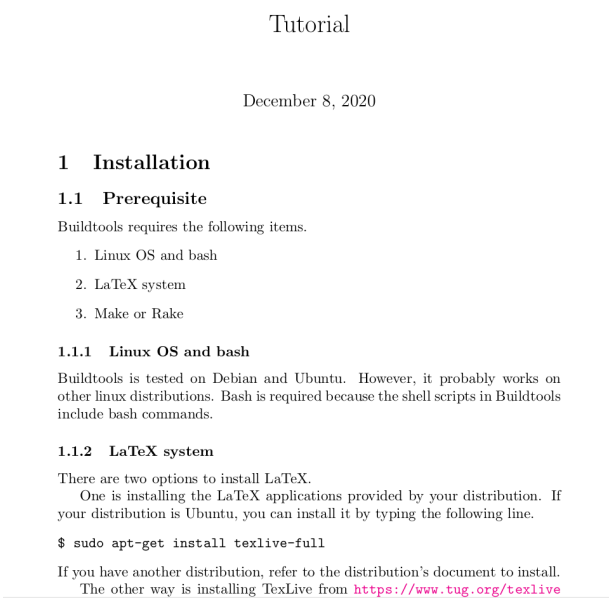
The first section and subfile are 'Installation' and installation.tex respectively. Maybe you edit a part of the section and test-compile to see how it looks like in the pdf file. Usually we come and go between editing and test-compiling repeatedly.

If the document is not so big, using rake is the best to test-compile, because it doesn't take much time and the pdf shows the whole document. This tutorial is rather a small document, so using rake for test-compiling is fine.

```
\subsection{Prerequisite}
Buildtools requires the following items.
\begin{enumerate}
\item Linux OS and bash
\item LaTeX system
\item Make or Rake
\end{enumerate}
 ... ...
 ... ...
```

Then, type the following line to see the pdf.

```
$ rake
$ evince Tutorial.pdf
```



Tutorial

December 8, 2020

1 Installation

1.1 Prerequisite

Buildtools requires the following items.

1. Linux OS and bash
2. LaTeX system
3. Make or Rake

1.1.1 Linux OS and bash

Buildtools is tested on Debian and Ubuntu. However, it probably works on other linux distributions. Bash is required because the shell scripts in Buildtools include bash commands.

1.1.2 LaTeX system

There are two options to install LaTeX.

One is installing the LaTeX applications provided by your distribution. If your distribution is Ubuntu, you can install it by typing the following line.

```
$ sudo apt-get install texlive-full
```

If you have another distribution, refer to the distribution's document to install.

The other way is installing TexLive from https://www.tug.org/texlive

5 Test compile

If you make a big document, for example a book which has more than 100 pages, then using rake is not a good way to test-compile. The bigger the dicument is, the longer time the compiling takes.

It is a better way to use 'lb' to test-compile a subfile separately. Lb makes a temporary rootfile which includes only the subfile and compile it once. The advantage of this way is very quick to compile. However, it has disadvantages. It only compiles the subfile, so the pdf you get is not a finished document. It compiles once, so cross-reference doesn't work at all. It is difficult to say which

is better. it depends on the size of your document. If it is very big, use lb to test-compile separately.

In this section, I want to show you how to use lb to test-compile a subfile. This document 'Tutorial' is not big, but it's OK. This is an example to show how to use lb.

Type the following.

```
$ lb installation
```

The argument is a subfile. The suffix can be left out. Then, lb makes a temporary rootfile '_build/test_installation.tex'. Its preamble is a copy of the preamble in the original rootfile. It has \input command and includes the subfile 'installation.tex'. Lb compiles '_build/test_installation.tex' and run the previewer specified in 'lb.conf' to show the pdf.

0.1 Prerequisite

Buildtools requires the following conditions.

1. Linux OS and bash
2. LaTeX system
3. Make or Rake

0.1.1 Linux OS and bash

Buildtools is tested on Debian and Ubuntu. However, it probably works on other linux distributions. Bash is required because the shell scripts include bash commands.

0.1.2 LaTeX system

There are two options to install LaTeX.

One is installing the LaTeX applications provided by your distribution. If your distribution is Ubuntu, you can install it by typing the following line.

```
$ sudo apt-get install texlive-full
```

If you have other distribution, refer to the distribution's document to install.

The other way is installing TexLive from https://www.tug.org/texlive Refer the documentations in the web to install TexLive system.

Lb compiles a subfile with the option synctex on. Therefore, if you open the source files, '_build/test_installation.tex' and 'installation.tex' in this example, you can do forward search and backward search between the source and pdf. If your use gedit and evince, backward search works by clicking left button with pressing down CTRL key, but forward search from 'installation.tex' doesn't work. If you want to do it, add the following line at the beginning of the subfile.

```
% mainfile: _build/test_installation.tex
```

However, forward search isn't used so often compared to backward search. Adding the line above is usually unnecessary.

6 Preprocessing

Sometimes, you want to do something before compiling your tex source files. For example,

- Some graphic files are generated with a program like gnuplot.
- Some tex files are generated with a program like pandoc.

Here in this tutorial I want to show you how to use pandoc as a preprocessing program. Pandoc is a document converter. It supports many types of format like markdown, latex, html and pdf. In this tutorial, 'Readme.md' in the Buildtools source files is converted to 'readme.tex', which is a latex source file.

Most distributions have a pandoc package, so you can install it easily. If your distribution is ubuntu, type

```
$ sudo apt-get install pandoc
```

To generate 'readme.tex', type

```
$ pandoc -o readme.tex ../Readme.md
```

You need to do two more things. One is changing main.tex and the other is changing helper.tex

First, add \input command to include readme.tex at the end of main.tex.

```
\documentclass{article}
\input{helper.tex}
 ... ...
 ... ...
\section{Make tarball}
  \input{tarball.tex}
\input{readme.tex}
\end{document}
```

In addition, uncommnet the eighth line to let \tableofcontents command to work.

```
 ... ...
\maketitle
% If you want a table of contents here, uncomment the following
 line.
\tableofcontents
 ... ...
```

Second, \tightlist command needs to be defined. Helper.tex is the best place to define it.

```
 ... ...
 ... ...
```

```
\providecommand{\tightlist}{%
  \setlength{\itemsep}{0pt}\setlength{\parskip}{0pt}}
 ... ...
 ... ...
```

This code is extracted from https://github.com/jgm/pandoc-templates/blob/master/default.latex.

Now you can compile it with rake.

```
$ rake
```

Contents

Now the contents of 'Readme.md' appears as the section nine in the table of contents.

We ran pandoc by hand in this section. If Readme.md is upgraded, we need to run pandoc again. It is a tiresome work and it should be done automatically. One good way is modify Rakefile so that rake does the preprocessing work automatically before compiling. It will be shown in the next section.

7 Use rake

Rake is a build tool similar to make. Rakefile describes instructions for rake to build source files. You can write any ruby commands in Rakefile. Therefore, it

has a high ability to describe the build process even if it is complicated.

Newtex generates a Rakefile, which is enough to compile the source files if there is no preprocessing procedure. In the previous section, we used pandoc to generate readme.tex. So, we need to modify Rakefile to put in pandoc. Modify the Rakefile as follows.

```
require 'rake/clean'

# if readme.tex doesn't exist, generate it first.
# This is necessary because readme.tex is accessed by gfiles in
#  line 12.
if File.exist?("readme.tex") == false
  sh "pandoc -o readme.tex ../Readme.md"
end
# use Latex-BuildTools
@tex_files = (`tfiles -a` + `tfiles -p`).split("\n")
@tex_files <<= "readme.tex"
@graphic_files = []
@tex_files.each do |file|
  @graphic_files += `gfiles #{file}`.split("\n")
end

task default: "Tutorial.pdf"

file "Tutorial.pdf" => "_build/main.pdf" do
  sh "cp _build/main.pdf Tutorial.pdf"
end

file "_build/main.pdf" => (@tex_files+@graphic_files) do
  sh "lb main.tex"
end

file "readme.tex" => "../Readme.md" do
  sh "pandoc -o readme.tex ../Readme.md"
end

CLEAN << "_build"
task :clean

task :ar do
  sh "arl main.tex"
  sh "tar -rf main.tar Rakefile"
  sh "gzip main.tar"
  sh "mv main.tar.gz Tutorial.tar.gz"
end
```

```
task :zip do
  sh "arl -z main.tex"
  sh "zip main.zip Rakefile"
  sh "mv main.zip Tutorial.zip"
end
```

Thanks to this modification, you don't need to run pandoc by hand. What you need is just type 'rake'.

There are websites about ruby and rake. For example,

- https://www.ruby-lang.org/en/
- http://rubylearning.com/
- https://ruby.github.io/rake/

8 Make tarball

You might want to distribute your source files. Then, you need to archive them. The script 'arl' looks for the subfiles and the graphics files included by the rootfile, and archive them.

- If -g option is given, it makes gzip compressed tarball.
- If -b option is given, it makes bzip2 compressed tarball.
- If -z option is given, it makes zip file.
- If no option is given, it makes non-compressed tarball.

If some latex source files are generated by preprocessing, you need to generate them before running arl.

```
$ arl
$ tar -tf main.tar
main.tex
edit_tex_files.tex
generate_templates.tex
installation.tex
lb.tex
preprocessing.tex
rake.tex
readme.tex
tarball.tex
test_compile.tex
helper.tex
Tutorial_1.png
Tutorial_2.png
hellolatex.png
```

```
tableofcontents.png
test_installation.png
```

Rakefile needs to be added to the tarball.

```
$ tar -rf main.tar Rakefile
```

Then, compress it into gzip.

```
$ gzip main.tar
```

The procedure above is already written in the Rakefile. Type 'rake ar', then rake makes a tarball.

```
$ rm main.tar.gz
$ rake ar
arl main.tex
tar -rf main.tar Rakefile
gzip main.tar
mv main.tar.gz Tutorial.tar.gz
```

Now the name of the tarball is 'Tutorial.tar.gz'.

If you want to make a zip file, type 'rake zip'.

The tutorial finishes at this section. Next section is the copy of Readme.md in Buildtools source files. It describes the background of Buildtools and features of each script.

9 Buildtools

9.1 The background of Buildtools

Buildtools is a part of Latextools If you make a long document, for example, a book with more than a hundred pages, you need to consider various things which isn't necessary in creating a short document.

- Divide the source file into small parts
- Compile each parts independently
- Replace something with another in the whole document
- Preprocessing (processes before compiling latex source files)

Latextools support these things and it includes two parts.

- Buildtools. It is tools which support creating templates of source files, building and partial compiling.
- Substools. It is tools which perform replacements in the whole document.

Buildtools is a part of Latextools and also a core tools in it. This document describes Buildtools only.

Dividing a source file Latex source files are simply called source files here. It is not appropriate to write a long document into a single source file. Because, bigger the file, much more difficult to edit it. To solve this, the document will be divided into some parts. Usually, they consist of a file containing `\begin{document}` and `\end{document}` and the other files called by the file with `\include` or `\input` command. The former file is called rootfile and the latter files are called subfiles. There is a difference between `\include` and `\input` though both are commands to include subfiles.

- `\include` can't be nested. it can be described only in the body, which means the part between `\begin{document}` and `\end{document}`. `\includeonly`, which must be in the preamble, specifies a list of files to include by the `\include` command. The files in the list are included by `\include` command, and files out of the list aren't included even if it is an argument of `\include` command. `\include` command issues `\clearpage` command before and after including the file.

- `\input` command simply include files. It doesn't issue `\clearpage` command. This command can be nested.

It is called build to make a document by compiling. It includes not only the compilation with latex but also the preprocessing, such as the image generation by gnuplot or tikz graph generation with data. It is finally completed by the compilation of the rootfile.

There are several programs to compile LaTeX source files, and they are called engines. Buildtools supports latex, platex, pdflatex, xelatex and lualate as engines.

The bigger the document to compile is, the longer the time needs. Even if you change a small part of the document, it needs the same time as the big change. You often need to compile to check how the pdf document looks like, which is sometimes called test, it needs long time in each compilation. The bigger the document is, the more serious the problem is. So, it has been thought up to compile a subfile itself without rootfile or other subfiles.

- Comment out the files in the argument list of `\includeonly` command which you don't want to compile.
- Use subfiles package.

Subfiles package is nice and many people recommend it. However, you need to include the package and use its command appropriately. Naturally, it is not the matter compared with the compilation time above.

Another way is to add an specific preamble to compile a subfile without any other files. More specifically, the subfile is put between "the text from `\documentclass` to `\begin{document}`" and "`\end{document}`". On that occasion, the subfile itself isn't changed, but another file, which contains the preamble, `\end{document}` and `\input` command, is made. The `\input` command reads the subfile. The file newly made is called "temporary rootfile". On the

other hand, the rootfile is sometimes called " original rootfile". The preamble in the temporary rootfile is a copy of the preamble in the original rootfile. The good point of this way is:

- There's no need to include any packages or put any special commands.
- Therefore, users don't need to install any packages when the source file is distributed.

The point is subfile can be compiled separately without any modification in the source files. The generation of the temporary rootfile is the only necessary. Buildtools has `ttex` shell script to do that.

Repeating compiling It often needs to compile source files two times or more because of the cross-reference. The repeating times are maybe two or three (or more), but I don't know the details. However, there's a great software that calculate the repeating times automatically. It is latexmk. Latexmk make the build very easy. Buildtools uses latexmk to compile rootfiles.

Build directory Some people might complain about latex because it generates various of auxiliary files and log files. If you make a build directory and put all the generated files into it, the source directory can be kept clean. One of such program is cluttex and it is recommended to people who like cleanliness.

Buildtools makes a temporary directory, which is also called build directory, and put all the temporary files and generated document. The default name of the directory is `_build`. It makes the source directory keep clean. If you want to see log or auxiliary files, search the build directory for them. It's very simple. Although it is superfluous to say so, meson build system which is very popular as a C build tool also uses build directory. This shows us that separating build directory from source directory is very easy to understand.

Lb, one of the tools in Buildtools, generate a final pdf document in the build directory. However, many users probably want to put it in the source directory. It is a natural idea. If you want to do so, use make or rake.

Rake is a similar program to make. Its advantage is using ruby language in the Rakefile, which is the script file of rake. Because ruby language is very strong and flexible, the script file can be readable and structured.

To get back to the subject how to put a final pdf document in the source directory, you just write a cp command in your Makefile or Rakefile to copy the pdf document under the build directory into source directory. Furthermore, the advantage to use make or rake is that it's possible to specify preprocessing code in the Makefile or Rakefile. Because preprocessing depends on the document and what tools the user choose, it's difficult for Buildtools to cover the preprocessing. Compared with that, Makefile or Rakefile are really flexible so that you can write your own preprocessing in it.

It is recommended that users should use make or rake with Buildtools.

combination with Texworks Lb, one of the tools in Buildtools, can either compile a rootfile or test-compile a subfile. If you specify it into proccessing tools in the texworks dialog, you can run lb from texworks and it's so convenient. Click edit, preference, then typesetting tab. Click plus button in the processing tools part, then put `lb`, `lb`, `$fullname` in name, program and arguments box respectively. Once you set it, you can compile a rootfile or test-compile a subfile by clicking on the typesetting icon (green triangle icon).

9.2 Buildtools structure

Buildtools is made up of the following five parts.

- `newtex`: It generates source file templates. This is used at the beginning of making documents.
- `lb`: It calls latexmk or ttex to compile source files.
- `arl`: It makes an archive file.
- installer
- a group of utilities. They are used by the tools above.

The following shows the steps to build documents.

1. Make the structure, especially the chapters/sections, of the document..
2. Run newtex and make folders and templates of the document.
3. Modify the templates of Makefile, Rakefile, cover page (cover.tex) and preamble (cover.tex) if necessary.
4. Write the body of the document and test-compile.
5. If necessary, make script files for the preprocessing.
6. Compile the rootfile to generate the final pdf document.

The step three to five above are usually repeated and the process is not necessarily in the order above.

9.3 Main tools

Each tool shows its help message if it is run with `--help` option. For example, newtex shows the following message.

```
$ newtex --help
Usage:
  newtex --help
    Show this message.
  Newtex.conf needs to be edited before running newtex.
  newtex
    A directory is made and some template files are generated under the directory.
```

The document of each tool is:

- The help message shown by each command with `--help` option.

- The description below in this document.

No other document exists. If you want to know more, see the source code. All the tools in Buildtools are shell scripts. If you are familiar to shell scripts, you can easily understand them because they are short.

newtex

```
$ newtex
```

Newtex is used when you make a new latex document. irst, decide the structure and chapters and make `newtex.conf` in advance. This script makes a directory and generates template files according to `newtex.conf`.

1. There is `newtex.conf` file in the Buildtools source files. Modify it to fit your environment and tex source files you will make.
2. Execute `newtex`. Then it make a directory which name is the same as `title` in `newtex.conf`. However, the space characters in the value of `title` is converted to underscore in the name of the directory. The script also generates template files under the directory.

lb

```
$ lb [LaTeXfile]
```

If the argument is left out, `lb` behaves as if `main.tex` is specified as an argument. `Lb` is a script to build LaTeX source files and you usually don't need anything except it.

- If the argument is rootfile, then `lb` compiles it with `latexmk`. If the argument is subfile, then `lb` runs `ttex` specifying the subfile as an argument.
- If the argument is rootfile, `lb` compiles it without synctex.
- If the argument is subfile, `lb` runs `ttex` and `ttex` compiles the subfile with synctex. After compilation, `lb` runs previewer specified in `lb.conf`.
- If there exists `lb.conf` in the current directory (it usually contains the rootfile), `lb` reads it and initialize some variables.
- If the variable `engine` in `lb.conf` is null string, then `lb` guesses an appropriate engine by itself. However, it is recommended that you should specify the engine in `lb.conf`.

You can specify the default values in `lb.conf` to initialize some variables.

```
rootfile=main.tex
builddir=_build
engine=
latex_option=-halt-on-error
preview=texworks
```

- `rootfile` is the name of the rootfile. If you specify the name of the rootfile as an argument to `lb`, then the argument takes precedence.
- `builddir` is the name of the build directory. Auxiliary files and output files are put in the build directory. If the argument of `lb` is a subfile, then the temporary rootfile is also put in the build directory. If `builddir` is empty, then no temporary directory is generated and the source file directory becomes a build directory.
- `engine` is the name of a latex engine. `latex`, `platex`, `pdflatex`, `xelatex` and `lualatex` can be specified. Other engines are not supported.
- `latex_option` is a list of options to specify as an argument to the engine through `latexmk`. `-output-directory` is automatically given to `latexmk` by `lb` even if `lb.conf`doesn't exist.
- `preview` is the name of a pdf previewer to show the document. It is run only if the argument of `lb` is a subfile.

arl $ arl [-b|-g|-z] [rootfile]

The name `arl` comes from "ARchive LaTeX files". It searches the rootfile for its related files (refer the following) and make an archive file. If the argument is left out, then it runs as if `main.tex` is specified as an argument.

- If there are preprocessing programs, you need to execute them before running arl.
- Arl archives the latex source files and the graphic files included by `\includegraphics` command.
- Therefore, `Makefile` or the preprocessing script files are not archived.

It is a good way to make a target (for example, its name is 'ar') in Makefile and write a recipe to add Makefile and the preprocessing files into the archive file made by `arl`. You can do the same thing with Rakefile.

There are options `-g`, `-b` and `-z` to compress the archive file into `tar.gz`, `tar.bz2` and `zip` file respectively. If no option is given, `arl` just make a tarball without compression.

Utilities You don't need to read this subsection except maintaining the scripts.

```
$ srf subfile
```

This script `srf` searches for the rootfile of the given subfile and outputs its absolute path. `srf` comes from "Search for Root File".

```
$ tfiles [-p|-a|-i] [rootfile]
```

It outputs a list of subfiles of the given rootfile. If the argument is left out, then it is run as if `main.tex` is specified as an argument.

- No option: It outputs a list of subfiles, specified from `\begin{document}` to `\end{document}`. They are the arguments of `\include` or `\input` command.

- `-p`: It outputs a list of subfiles specified with the `\input` commands in the preamble of the rootfile.
- `-a`: It outputs a list, outputted with no option, and the rootfile itself.
- `-i`: It outputs a list of subfiles specified with both `\include` and `\includeonly` command.

The files in the list are separated with new lines.

```
$ tftype [-r|-s|-q] files ...
```

It outputs or returns the type of the given latex files.

- `-r`: It outputs rootfiles which is in the argument files. If no options are given, It behaves as if this option is specified.
- `-s`: It outputs subfiles which is in the argument files.
- `-q`: It doesn't output anything. The number of the argument must be one. It returns an exit status code of the file type. If the code is 0 or 1, then the file is a rootfile or subfile respectively. Otherwise an error happens.

Using `-q` option is the most common.

```
$ gfiles files ...
```

The argument is a list of latex source files. It outputs graphic files which are included by `\includegraphics` commands in the given files.

```
$ ltxengine rootfile
```

It guesses a latex engine to compile the rootfile, although it is recommended that the engine should be specified by the user. For example,

```
\usepackage[luatex]{graphicx}
```

If this command exists in the preamble, it guesses that the engine is probably lualatex.

```
$ ttex [-b builddir] -e latex_engine [-p dvipdf] [-v previewr] -r rootfile subfile
```

It generates a temporary rootfile of the subfile and compile it. The compilation is done only once. Therefore, no cross-reference is carried out. This comes from the idea that `ttex` is a script for test to see how the pdf file looks like and cross-reference is not so important. The cross-reference to the other files doesn't work neither. This script can be run directly on the command line, but usually it is called by `lb`. The following shows the options.

- `-b`: Specify a build directory. The default is `_build`.
- `-e`: Specify a latex engine. There is no restriction on the engine, but it assumes that latex, platex, pdflatex, xelatex or lualatex is specified.
- `-p`: Specify an application that translates dvi into pdf. This is used only when the engine is latex or platex, because they outputs a dvi file. The default is dvipdfmx. Other possible application is dvipdfm and dvipdf.
- `-v`: Specify a pdf previewer like evince. If you edit the source file with texworks, it is a good idea to specify texworks here.
- `-r`: Specify the original rootfile.

9.4 Installation and uninstallation

Prerequisite

- Linux and bash: It is tested on Debian and Ubuntu. However, it probably works on other linux distributions. Bash is required because this script includes bash commands.
- LaTeX: There are two options to install LaTeX. One is installing the LaTeX applications provided by your distribution. The other is installing TexLive.
- make or rake: These applications are not necessarily required to run the tools in Buildtools. However, it is recommended that they are used under the control of make or rake. You don't need to install both of them. Choose one which you like. Make is a traditional build tool originally aimed at C compiler. Rake is a build tool similar to make. It is one of the ruby application. The advantage to use rake is that you can put any ruby codes into Rakefile, which is the script file of rake. Generally speaking, Rakefile is easy to understand than Makefile.

Installation Use the script install.sh.

```
$ bash install.sh
```

This script installs the executable files into `$HOME/bin`. Debian and Ubuntu adds the directory `$HOME/bin` into `PATH` environment variable if it exists at the login time. The script makes the directory `$HOME/bin` if it doesn't exist. In that case, you need to re-login to put the directory into the `PATH` environment variable. If you run `sudo` or `su` to become the root user before the installation, the executable files are put into `/user/local/bin`.

If your OS is debian, type the following to be the root user.

```
$ su -
# bash install.sh
```

Or if it's ubuntu,

```
$ sudo bash install.sh
```

Uninstallation Use the script uninstall.sh.

```
$ bash uninstall.sh
```

If you run this as a user, the files under `$HOME` are removed. If you run this as a root, the files under `/usr/local` are removed. If your OS is debian, type the following to be the root user.

```
$ su -
# bash uninstall.sh
```

Or if it's ubuntu,

```
$ sudo bash uninstall.sh
```