C-ALE Presentation Prep Manual
VERSION 0.1

Jerry Cooperstein

06/11/2018

Note: This document is for use by speakers at the C-ALE workshop at OSSCON-NA in Vancouver, September 2018. Please ask for permission before using for any other purpose.internal to the

This documentation is not meant to be fully comprehensive, or to teach the use of either **git** or **LaTeX**. It is really more of an FAQ.

Contents

1 Basic Ingredients

1. For now it is best to work in a directory named CALE. We may adjust this later to something different so we don't have to rename the final results.
2. You need to use the LFS301/common/ subdirectory which contains the Makefile as well as LFS301/common/texmf which contains all LaTeX style and class files. Like the Makefile these are also shared by all talks and should be handled with care. They are in a separate git repository.
3. There must be a master .tex file currently named CALE.tex. All other content source files will be included from there.
4. The preferred mode for including chapter by chapter (or subtopic subtopic) material is to put it in subdirectories under CHAPS, with a master index.tex in each directory including files corresponding to individual sections. Doing it this way makes reordering or including/excluding individual chapters and sections trivial, and numbering is automatic. You are givin a sample CALE.tex file and a couple of subdirs under CALE to show how it works.

2 Build Procedure

The only two output files authors really need to produce and review and make sure they look fine are:

1. CALE.pdf

   The book students will receive, produced by doing:

   ```
   $ make
   ```

2. CALE-SLIDES.pdf

   The slide deck that the instructor teaches from, produced by doing:

   ```
   $ make slides
   ```

Note that each page of the book containes the slides that are projected, plus additional notes etc on the bottom half of the page; i.e., CALE-SLIDES.pdf is included in CALE.pdf.

Most of the other **make** targets are for the use of final packaging before it goes to the printer, which is not a task for the course maintainer. You can see a (hopefully complete) list of targets by doing:

```
$ make help
make TARGETS:
none or class:  Produces CALE.pdf (full-size, color)
slides/SLIDES:  Produces CALE-SLIDES.pdf (slide-size, color)
outline:        Produces CALE_{short,long}_outline.tml (do make first)
COVERS          Produces full, front and slides covers
RELEASE/release Populate RELEASE directory with all PDFS with version numbers embedded
spellcheck      Check spelling
spellcheck-list-files  List files that would be spell-checked
FINDOVERFULL    Find vertical and horizonal overruns
help HELP or usage: Produces this message
```

3 Git

All authors will be sharing the same two repos. Please avoid working on the master branch in either repo. After you have given us your github ID and gotten the inivitation, you can obtain them by doing

```
git clone git@github.com:/cooplf/CALE
cd common
git clone
git@github.com:/cooplf/CALE-common
```

These are private github repos. You don't have to do the common repo in the common subdir, (you can have a link to somewhere else) but it is easiest to do that.

We are sure all of you are git-literate, but here is a primer of the best way to do this:

We ask you to **not work on the `master` branch** of `CALE` and `CALE-common`, and leave that to the content chief. You should work on one or more of your own branches, which you can create and switch to as in:

```
$ git branch boris
$ git checkout boris
```

and work to your heart's content. It's best to make many small commits as in

```
$ git add -v -u
$ git add -v .
$ git add -v  file 1 file 2 somedir ...
$ git commit -m"some message"
$ git push -v
```

(Note the first time you push your git branch to the repo it may warn you it doesn't exist yet and give you the proper command to do so.)

I often do:

```
$ git commit -a -m"some message"
```

which does the add and commit at the same time, but will only pick up files which already existed, not new ones.

Please **push often** even if you are the only one you expect to look at your work in progress. For one thing it gives you an online backup, and it does permit the head content chief to look at things so that he can minimize conflicts if he is also working on the repo, say on the infrastructure, which is always been modified and improved.

Please remember, all **git** repo collaborators are equal! You can obliterate, modify, etc the `master` branch as well and the head of content. So PLEASE exercise caution and stay away from `master`.

(If this was a long term setup we would be smarter and be pulling from contributor branches etc., but for this small project this workflow is both sufficient and easy.)

4 Content: Main .tex file

In the main directory you need a rather simple file, whose content we will explain. Here is the template in the master repo as an example:

```
\documentclass{CALE}
\newcommand{\course}{CALE}
\newcommand{\version}{3.14159}
\newcommand{\speaker}{Bullwinkle J Moose}
\newcommand{\coursetitle}{An Interesting Talk\\ by\\ Bullwinkle J Moose}

\begin{document}

\topshit
```



```
\subimport{CHAPS/prelim/}{index}
\subimport{CHAPS/linux-concepts/}{index}
\end{document}
```

For your talk you obviously change the title, course number, and the version and name the speaker, and include the proper chapters. That's it!

Note the use of the `\subimport{}` command, rather than `\include{}`, which you might expect if you are used to using LaTeX. This is rather subtle, but it permits things in the CHAPS subdirectories to use `\include{}` without giving full paths.

5 Content: sections in CHAPS

Each subdirectory in CHAPS represents a section in the course. Here is a one from the sample:

```
$ ls -l CHAPS/prelims
total 32
drwxrwxr-x 2 coop coop 4096 Mar 27 17:58 ./
drwxrwxr-x 4 coop coop 4096 May 29 14:00 ../
-rw-rw-r-- 1 coop coop 4051 Mar 27 17:58 cline.tex
-rw-rw-r-- 1 coop coop 5949 Mar 27 17:58 dists.tex
-rw-rw-r-- 1 coop coop  110 Mar 27 17:56 index.tex
-rw-rw-r-- 1 coop coop 3576 Mar 27 17:56 labs.tex
-rw-rw-r-- 1 coop coop 1380 Mar 27 17:56 sudo.tex
```

The index.tex file should be in each chapter subdirectory and looks like:

```
\chapter{Preliminaries}

\lflogo

\minitoc

\clearpage
\input{cline}
\input{sudo}
\input{dists}

\input{labs}
```

The chapter (session) title goes here with the `\chapter{}` macro.

Each of the files referred to by `\input{}` is a section and starts with a `\section{}` macro as in:

```
\section{The Command Line}
```

(We will look at the rest of what is in these files shortly.) note you do **not** have to use the `.tex` extension when including, so `\include{yum}` is the same as `\include{yum.tex}`.

This format makes it easy to slip in new subsections or remove or reorder without error-prone cutting and pasting in large files. It also keeps directory sizes small.

A file containing lab exercises for this section can be called either lab.tex or labs.tex. (You do not have to have a lab file.) It has a somewhat different content than the other sections and we will discuss this separately.

6 Frames, Slides, and Notes

Each (non-lab) section is composed of a serious of **frames**, each of which contains a **slide** followed by **notes**.

- LFS301-SLIDES.pdf contains only the slides and is what the instructor projects.
- LFS301.pdf contains on each page the slide followed by the notes; it is exactly what is in the book each student receives.
- We do not use the well-known **beamer** extension to LaTeX even though we use frames etc. as it is a mess to use for our purposes.

Each section starts with a `\section{}` title macro, and then is a sequence of **frames**; nothing else goes in these files. Some subsections have only one frame, some have many. Each section will appear in both the master table of contents and the smaller one at the start of each section.

Each frame/note couplet has the structure:

```
\begin{frame}
{frame title}
.........
\end{frame}

\cprotect\note{
....
   }
```

Here is an example of a section file with only one frame:

```
\section{Daemons}

\begin{frame}
   {Daemons}

   \begin{itemize}

      \item A \textbf{daemon} is a background process whose sole
      purpose is to provide some specific service to users of
      the system.
      \item Can be quite efficient because they only operate
      when needed.
      \item Many are started at boot time.
      \item Names often (but not always) end with \textbf{d}:
      Examples include \textbf{httpd} and \textbf{systemd-udevd}.
      \item May respond to external events (\textbf{systemd-udevd}) or
      elapsed time (\textbf{crond}).
      \item Generally have no controlling terminal and no
      standard input/output devices.
      \item Sometimes provide better security control.
   \end{itemize}

\end{frame}

\cprotect\note{

   A daemon process usually is a background process that
   provides a specific service to users of the system.  It's
```

```
   sole purpose on the system is to provide that specific
   service.  Some examples include \textbf{xinetd},
   \textbf{httpd}, \textbf{lpd}, \textbf{telnet}, and
   \textbf{vsftpd}.

directory.  }
```

> **Note:** You should have a `\note{}` section even if it empty. It won't affect the slide deck if missing, but it may screw up the book depending on placement.

Almost all frames contain itemized lists as in the above example.

Frames may also contain graphical images with `\includegraphics[]{}` with or without text. They may also include tables using `\begin{longtable}`. Both should have **captions**. We will discuss later.

To repeat, what is in the top of the frame is what is in the instructor slide deck. The notes section appears only in the book.

7 LABS

Each section may have a labs.tex (or lab.tex) file.

This file should look like:

```
\begin{Lab}

   \begin{exe} Using \textbf{wget}

      \begin{enumerate}
         \item
         Use \textbf{wget} to download the web page at
         \url{https://www.kernel.org}.
         \item
         Use a text-based web-browser to look at the downloaded
         file (probably named \filelink{index.html}).
         \item
         Use a graphical browser to look at this file.
      \end{enumerate}
      \begin{sol} \

         \begin{enumerate}
            \item
            \begin{raw}
$ wget http://www.kernel.org
   \end{raw}
   \item
   For a text-based browser do:
   \begin{raw}
$ lynx index.html
   \end{raw}
   using the text-based browser on your machine.
   \item
   For a graphical browser you will need to point to
   \url{file://<exact absolute location>/index.html}.
```

```
Note that when you put in the absolute path to the
file you should see three slashes after \verb?file?
as in \filelink:{file:///tmp/index.html}.

\end{enumerate}
\end{sol}
\end{exe}

\begin{exe} Using \textbf{curl}
   ......
\end{exe}
\end{Lab}
```

Please note the following:

- You can use all the usual LaTeX syntax and commands.
- Always leave a blank line after `\begin{exe}` and `\begin{sol}`.
- Whatever is on the `\begin{exe}` line is the title of the lab.
- You do not have to have a solution (the material between `\begin{sol}` and `\end{sol}`.

8 Verbatim Methods

One thing that can be a bitch in LaTeX is dealing with **verbatim** situations, where you do not want things interpreted as LaTeX commands or mathematical symbols etc. For example you cannot use and display characters such as `$, _, {, }, #` without some care.

We are going to show you a number of ways to display such **verbatim** material, ranging from a character, to a string or a few lines, or entire files. Of the methods we usually use, only `\verb` and the `alltt` environment are actually native LaTeX methods; the others are special commands and macros defined in our style files. (Note: you can still use the usual verbatim environments of LaTeX as well of course, upon which our custom macros are built.)

The simplest method is the `\verb` method, which can be done inline without causing a line break, as in

```
\verb?$ command arg1 arg2 arg3?
```

Whatever is between the `?`'s will be rendered in raw or verbatim form. Note you can use other characters (like with **sed**) as in: `\verb:verbatim_text $ #:`. You can not have a `?\verb?...?` be split across a line.

More complicated multi-line things like code listings are easily done with the **raw** environment and **rawfile** macro, both of them custom for us, as we now describe.

First here is the use of the **raw** environment:

```
\begin{raw}
line 1
line 2 of verbatim_text
\end{raw}
```

rendering

```
line 1
line 2 of verbatim_text
```

which produces a smaller font (desirable) than the more fundamental LaTeX environments on which it is based, and also gives it a blue color.

Occasionally you will want to use a smaller font in order to avoid ugly line wrapping, so you have the following variations:

```
\begin{rawfootnotesize}
line 1
line 2 of verbatim_text
\end{rawfootnotesize}
```

```
line 1
line 2 of verbatim_text
```

```
\begin{rawscriptsize}
line 1
line 2 of verbatim_text
\end{rawscriptsize}
```

```
line 1
line 2 of verbatim_text
```

(Note, `\raw{}` is the same as `\rawsmall{}`)

If for some reason you want a **HUGE** size:

```
line 1
line 2 of verbatim_text
```

```
line 1
line 2 of verbatim_text
```

If you want to include an entire file verbatim, such as a program source or a script there are two basic LaTeX macros:

```
\VerbatimInput{script.sh}
\verbatiminput{file.c}
```

but please use our custom macro:

```
\rawfile{file.h}
```

which is preferred. Why? Three reasons:

1. It handles finding the path of the input file properly without having to make it absolute.
2. It uses a smaller (blue) font which is good.
3. It is extensible

There are also the following size variations you can use

```
\rawfilesmall{}
\rawfilefootnotesize{}
\rawfilescriptsize{}
\rawfiletiny{}
```

(Note, `rawfile{}` is the same as `rawfilesmall{}`)

Finally there is one more builtin environment **alltt**, which we have redefined to use smaller (blue) fonts. It differs from other verbatim environments in that LaTeX commands are recognized inside the verbatim environment, so you can do something like:

```
\begin{alltt}
\$ tar jxvf /home/student/LFT/\course/SOLUTIONS/s_\thesection/u-boot_2011.03.tar.bz2
\end{alltt}
```

Why do this? In this case the `\course` macro and the `\thesection` macro would not fill in the final text which will look like:

```
$ tar jxvf /home/student/LFT/LFD411/SOLUTIONS/s_04/u-boot_2011.03.tar.bz2
```

Furthermore, you have to properly escape the special characters in the above (`$` and `_`) as (`\$` and `\_`) as otherwise LaTeX will fail to compile to a pdf without it. Furthermore if you want a output of more than one line you have to explicitly insert a `\newline`:

```
\begin{alltt}
line 1 \newline
line 2 \newline
line 3
\end{alltt}
```

9 Hyperlinks

The simplest way to do a hyperlink is with: `\url{http://www.linuxfoundation.org}` which is the proper way to do internet links.

However this won't work properly on links on the local machine. For this we have a custom macro: `\filelink{/etc/passwd}`.

> Note this will fail on files without extensions and results will vary according to what pdf browser you use – **evince** seems to do better than **acroread** for instance. Customizing exactly what app opens when you hit a link is enough to make you pull your hair out and we won't discuss it here.

Note that while `\url{}` will properly handles references with special characters, such as `_`, `\filelink{}` will not, so you will have to do something like `\filelink{/etc/some\_file.conf}`.

The underlying way to do links is with the basic LaTeX macro:

```
\href{link}{text to click on}
```

You can always do this if necessary but it won't solve all problems. and we don't recommend having the link being different than the highlighted text as it won't appear in the book properly.

10 Images

Images can be a number of different formats, including `.png`, `.jpg`, and `.pdf`. It is also possible to use `.svg` with a little care.

The proper way to insert an image is with:


```
\begin{figure}[H]
  \includegraphics[scale=0.60]{mono.png}
  \caption{Monolithic Kernel Architecture}
\end{figure}
```

If you omit the caption, the figure will not appear in the list of figures at the beginning of the manual.

If the `[scale=0.60]` is omitted the scale will default to 1.0. It is **much better** to leave images at full size and rescale them this way as LaTeX does a far better job quality-wise than most image manipulation software. You'll probably have to iterate on the scale.

Alternatively, you can use `[width=7.0in]` or `[height=xx.in]` etc.; you will have to play with values for appearance. For slides which contain only work nicely images, usually `height=3.3in` or `width=6.0 in` . You have a couple of choices for where to put the images. In the above example, they are in the same directory as the `.tex` file. Sometimes you want a directory containing all images for the class, this is an author decision. In this case you might have:

```
\begin{figure}[H]
  \includegraphics[scale=0.60]{IMAGES/mono.png}
  \caption{Monolithic Kernel Architecture}
\end{figure}
```

for example.

11 Tables

Tables are produced using the `longtable` environment. While you can use other LaTeX table environments, such as `tabular` or `tabularx`, captions will not work in those environments.

The following table:

Table 1: **printk() Log Levels**

macro	Level	Meaning
KERN_EMERG	0	system is unusable
KERN_ALERT	1	action must be taken immediately
KERN_CRIT	2	critical conditions
KERN_ERR	3	error conditions
KERN_WARNING	4	warning conditions
KERN_NOTICE	5	normal but significant condition
KERN_INFO	6	informational
KERN_DEBUG	7	debug-level messages

is produced by:

```
\begin{longtable}{|l|l|l|}
   \caption{printk() Log Levels}\\
   \hline
   \textbf{macro}       &  \textbf{Level}  &  \textbf{Meaning}   \\ \hline
   \verb?KERN_EMERG?    &  0  &  system is unusable              \\ \hline
   \verb?KERN_ALERT?    &  1  &  action must be taken immediately\\ \hline
   \verb?KERN_CRIT?     &  2  &  critical conditions             \\ \hline
   \verb?KERN_ERR?      &  3  &  error conditions                \\ \hline
   \verb?KERN_WARNING?  &  4  &  warning conditions              \\ \hline
   \verb?KERN_NOTICE?   &  5  &  normal but significant condition\\ \hline
   \verb?KERN_INFO?     &  6  &  informational                   \\ \hline
   \verb?KERN_DEBUG?    &  7  &  debug-level messages            \\ \hline
\end{longtable}
```

Note the use of `{|l|l|l|}` specifying the vertical lines and letting LaTeX figure out optimal column widths. Sometimes this is not the best choice, especially if lines have to wrap within columns. In that case you use instead (for example):

```
\begin{longtable}{|p{0.2\textwidth}|p{0.1\textwidth}|p{0.6\textwidth}|}
...
```

producing:

Table 2: **printk() Log Levels**

macro	Level	Meaning
KERN_EMERG	0	system is unusable
KERN_ALERT	1	action must be taken immediately
KERN_CRIT	2	critical conditions
KERN_ERR	3	error conditions
KERN_WARNING	4	warning conditions
KERN_NOTICE	5	normal but significant condition
KERN_INFO	6	informational
KERN_DEBUG	7	debug-level messages

This line defines the horizontal widths; generally they shouldn't add to more than 0.9 times the text width. You have to manually iterate until you get it looking right. The `|` is what produces the vertical lines.

12 Some Optional Custom Macros and Environments

The file CALE/common/texmf/tex/latex/LFALE/CALE.cls includes all necessary packages and defines the custom commands and environments used by LaTeX .

Here is a list of custom **environments** you might use:

- `lfbox`

  Used to put text in boxes such as:

  > – First item in the box
  > – Second item in the box

  which was rendered by:

```
\begin{lfbox}
  \begin{itemize}
    \item First item in the box
    \item Second item in the box
  \end{itemize}
\end{lfbox}
```

  Note that text in boxes does not look good if it is not in a list with bullets. In the future the style might be changed to put in a background color etc.