Linux Foundation Style Manual and Preparation Guide for Developer and Enterprise System Administration Courses Version 6.0

Jerry Cooperstein
Behan Webster

04/05/2021

Please Note

This document is internal to the **Linux Foundation** and those working with us on courseware. It is not to be distributed outside that arena, and is considered private.

This is a Reference Manual, Not a Tutorial

There is a lot of information here and you do not need to absorb it all to get working.

The best way is to look at an already functioning course source. Make sure you can build it and look at the files in it.

There are a lot of refinements in this document that are not necessary to get going, but like all such documents it began as a short summary and blossomed.

Two other documents you should receive as needed are a **Cheatsheet** with a lot of basic information in condensed form, and for those translation from English to other languages a **translation guide**: we have gone to great lengths to abstract out language dependences so all that is needed is a language-based branch in **git** with no changes in the build software.

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

Any pre-existing **LFD** class can be used as a template; when we need to be specific we have taken examples from **LFD420** to demonstrate the use of figures, tables, indexing, etc.

LFD and LFS Course Formatting Differences

LFD courses (i.e, Linux Foundation Developer training) and **LFS** courses (i.e., Linux Foundation Enterprise IT and Linux System Administration training) use the same infrastructure including Makefiles, style files, scripts as etc.

However they have a different look and feel because the **LFS** courses use a “slide + notes” presentation while **LFD** courses use more of a “book” format.

Many features, however, are common to both. Thus it is completely possible to mix and match both types of content together in custom courses.

Contents

1 Using LATEX

You may be either frightened or disgusted at using LATEX. You may consider it inferior to WYSIWYG software, or more modern (or at least recent, one can argue about what is modern) alternatives. So let me list some of the reasons we use LATEX:

- Longevity and support

  I first started using TEX, Amstex, and LATEX in the 1970s. It was invented by Donald Knuth (look him up if you don't know who he is) as a side project to scratch an itch.

  Despite its vintage, it is considered almost bug free and constantly is being improved and extended by a very large community of contributors. Almost anything you need to do can be researched quickly on **Google**.

- Available on all distributions and operating systems

  Occasionally there are some inconveniences, such as requiring too many packages to be installed you will never need, or in the case of **RHEL**, using either very old versions on **RHEL6** or missing some small files in **RHEL7** (which are supposed to be added to **EPEL** as I have been bitching about this ever since **RHEL7** came out.) But there are no problems I am not aware of (AFAIK).

- Does a better job of justification, line breaking, hyphenation etc, than any other document processing system; in fact other software has been trying to implement the algorithms developed for TEX which work on whole paragraphs as a unit, instead of line by line.

- Courseware maintainers are free to use any legal LATEX macros, environments, etc; they are not restricted to what we have used before. This often leads to enhancements made possible in every course.

- LATEX has very advanced indexing software, builds really good tables of contents, lists of tables and figures, etc. In the **LFS** courses we have not taken advantage of these features, but we do extensively in the **LFD** classes.

- Global searches, find and replaces etc, are very easy through standard tools like **grep, sed, perl** etc.

There are disadvantages, however:

- Errors are sometimes cryptic, and it can be hard to figure out exactly what induced them. Please do not hesitate to contact me when confused.

- it is not WYSIWYG. However, you can mitigate this by keeping the pdf displayed while you are editing and making. If you are using **evince**, the display will update every time it changes.

- The class and style files are rather dense and opaque to non-experts. However, you should not have to mess with them.

Installing LATEX from original media

While all **Linux** distributions have LATEXpackages, they vary by version and package choice, as mentioned earlier. A method which eliminates this variation is to use the original source media.

Doing this is pretty easy. Go to https://www.tug.org/texlive/acquire-iso.html and follow the instructions for getting an iso image file. You can also find installation instructions. Usually this amounts to something like

```
$ sudo    mount -o loop texlive2019.iso /mnt
$ cd /mnt
$ sudo ./install-tl
```

This takes some time and it will install (by default) about 6 GB under `/usr/local/texlive`. The only remaining thing to do is to put the associated programs in your `PATH` by putting

```
 PATH=/usr/local/texlive/2019/bin/x86_64-linux:$PATH
```

in your shell initialization file, `~/.bashrc`. Note the new directory is put **before** your current path in case you already have distribution packages installed, so they do not interfere with the native **texlive** install.

It is kind of wasteful to use this much space on your machine but it makes life easier in the long run.

2 Philosophy and Methodology

Very Important

This section is not for LFS courses This section is only relevant to **LFD *book style*** courses.

The conventional method used by many training outfits is to have the instructor present a series of slides with bullet points, either in a presentation program, or in html. These tend to have not much detail.

Accompanying this may be additional notes for the instructor's use, and a printed manual, which may or may not contain additional notes and go into more depth.

In practice, three problems arise:

- A lot of material may reside only in the instructor's brain, and is not readily available. This may work well when the instructor is the developer of the class, or someone who is very experienced with the material and has sat through an experienced instructor on the material teaching the class. But it is very hard for someone new tasked with delivering the class.
- The slides become decoupled from the more in-depth information, and every time there is an update both have to be modified. Often one or the other lags behind.
- The problem of producing a printed instruction manual for the students gets complicated. There tends to be a lot of manual work to do every time there is an update. Once again synchronization is difficult.

These problems are not insurmountable, but we have taken a different approach. In our method:

- The instructor should not get up there and just read the presentation material; but it is there to remind him/her of what to talk about it.
- The printed manual is closely coupled to the presentation material. The source is all done in LaTeX.

3 Git

There are two private github repos you need to deal with:

1. A repo specific to the course (e.g., `LFS300`)
2. A `common` repo used by all courses.

Once you have been given access permissions you should **clone** these repos as in:

```
$ cd LFCW
$ git clone -v git@github.com:/cooplf/LFS300
$ git clone -v git@github.com:/cooplf/common
```

The directory you clone from doesn't matter, we have used LFCW here and you can substitute whatever you want for it. We will now have two directories, LFS300 and common. Note that there is a symbolic link in LFS300 pointing to where common is; we do not recommend changing this.

Don't do this

Do not work on the master branch in either repo; 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  a_list_of_files
$ git commit -m"some message"
$ git push -v
```

Please Note

Note the first time you push your git branch to the repo it will 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.

You should not need a lot of **git** commands; usually you can get everything you need to do for this work with:

```
$ git clone
$ git checkout
$ git branch
$ git push
$ git pull
$ git add
$ git commit
$ git merge
$ git rebase
$ git gc
```

We cannot really give a **git** primer here, but authors not familiar with it pick up what is needed pretty quickly. If you are a GUI kind of person you can also use the **github.com** interface. Personally, I know nothing about GUIs used with **git**.

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`.

4 Basic Ingredients

There is no need to manually number chapters (sessions). LaTeX will take care of that. This makes it very easy to change the order of sessions, or insert new ones, or remove existing ones.

In addition LaTeX also automatically assembles its table of contents, list of figures, and list of tables. Indexing can also be done in a straightforward manner.

In what follows I will explain how to use this production system. Producing a course requries these necessary ingredients:

1. A directory with the same name as the course (e.g., `LFD459` or `LFS492`. Subsequent references are with with respect to that directory.
2. A master course file (`LFD459.tex`, `LFS492.tex`) which includes inputting the master **class** and **style** files, version information, and perhaps some various custom commands, and then includes a master file for each session of the class, which in turn includes subsection files.
3. A `CHAPS` or `MODULES` directory tree, with a directory for each session, containing a `index.tex` file enumerating the subsections, and also the subsection files.
4. For each directory that has laboratory exercises there should be a `labs` subdirectory under the session directory. This is where lab solutions (**C** files, scripts etc.) should go.

Don't do this

Note that one should **not** place large binary tarballs etc under the `labs` directory. They should point to something under a `RESOURCES` directory, as in

On the command line

```
c8:/teaching/LFCW>ls -lF LFS301/CHAPS/virtual/labs
total 0
lrwxrwxrwx 1 coop coop 45 Feb  8  2019 Core-current.iso ->
↪  ../../../../RESOURCES/LFS301/Core-current.iso
lrwxrwxrwx 1 coop coop 49 Feb  8  2019 CorePlus-current.iso ->
↪  ../../../../RESOURCES/LFS301/CorePlus-current.iso
lrwxrwxrwx 1 coop coop 49 Feb  8  2019 TinyCore-current.iso ->
↪  ../../../../RESOURCES/LFS301/TinyCore-current.iso
```

This is done to keep repo size down, the files in `RESOURCES` will be made available to students in a different way.

5. An `IMAGES` directory with all necessary graphics files.
6. A directory `SOLUTIONS` which will contain the solutions that will be distributed to students. Most classes have only a `LICENSE` and a `genmake` file, and a `Makefile` appropriate to the class; the rest is produced by `make SOLUTIONS`.

Thus, all source resides in the `CHAPS` directory tree. The steps for producing the course pdfs are:

1. Work in a directory named after the course (e.g.., `LFS300`).

2. This contains the `Makefile` that is common to all courses. (This is a link to `common/Makefile_onecourse`.) If you find it useful to patch it somehow, make sure you let me know as I periodically update the file in all courses, so I can accept or reject as a universal change.

3. Use the `LFS300/common/` subdirectory which contains necessary scripts as well as `LFS300/common/texmf` containing LaTeX style and class files. Like the `Makefile` these are also shared by all classes and should be handled with care.

4. There must be a master `.tex` file with the course name, i.e., `LFS300.tex` which we will explain in detail. All other content source files will be included from there.

5. The way to include chapter by chapter (really session by session) 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.

Don't do this

Avoid putting chapter and figure numbers in file names. When things get shuffled later this does nothing but create a complete mess of tangled spaghetti and is very error-prone. Use descriptive names instead.

(It is very common to reorder, add and delete chapters and subsections within chapters.)

There are other files in the main directory which are the same for each class as well. A typical example:

On the command line

```
$ ls -laF LFS300/
  total 188
drwxr-xr-x  6 coop coop 69632 Apr  6 16:10 ./
drwxrwxr-x 53 coop coop 24576 Apr  8 07:13 ../
-rw-rw-r--  1 coop coop   676 Apr  6 11:45 .aspell.en.prepl
-rw-rw-r--  1 coop coop 10977 Apr  6 11:45 .aspell.en.pws
-rw-rw-r--  1 coop coop  1215 May 16  2019 blurb.HTML
drwxrwxr-x 20 coop coop  4096 Oct  3  2017 CHAPS/
lrwxrwxrwx  1 coop coop     9 Jul 25  2017 common -> ../common/
drwxrwxr-x  8 coop coop  4096 Apr  6 14:24 .git/
lrwxrwxrwx  1 coop coop    17 Feb 19  2018 .gitignore -> common/.gitignore
drwxrwxr-x  2 coop coop  4096 Apr  6 09:36 IMAGES/
-rw-rw-r--  1 coop coop  1195 Apr  6 14:24 LFS300.tex
lrwxrwxrwx  1 coop coop    24 Jun 10  2016 Makefile -> common/Makefile_oneclass
-rw-rw-r--  1 coop coop    57 Mar 11  2019 NOTES
drwxrwxr-x  2 coop coop  4096 Apr  6 16:10 SOLUTIONS/
-rw-rw-r--  1 coop coop 39568 Apr  6 11:45 wordlist.dict
-rw-rw-r--  1 coop coop 10968 Apr  6 11:45 wordlist.txt
```

These contents are essentially the same for all courses. We will comment on some of these files later, and explain in detail the LaTeX content files.

5 Build Procedure

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

1. `LFS300.pdf` The book students will receive, produced by doing:

On the command line

```
$ make
```

2. `LFS300-SLIDES.pdf` The slide deck that the instructor teaches from, produced by doing:

On the command line

```
$ make slides
```

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.

Very Important

One exception is that for some translated courses the only files that need to be processed are the lab exercises, in which case the relevant commands are

On the command line

```
$ make labs
$ make splitlabs
```

which produces one file containing all the labs (`LFS300-labs.pdf`) and a directory (`LABSPLIT`) which contains each lab in a separate file.

This is also required in produced the labs for courses with an e-learning sibling. It is not necessary to discuss this here.

You can see a (hopefully complete) list of targets by doing:

On the command line

```
$ make help
make TARGETS:
none or class:  Produces LFS300.pdf (full-size, color)
slides/SLIDES:  Produces LFS300-SLIDES.pdf (slide-size, color)
outline:        Produces LFS300_{short,long,appendix}_outline.tml (do make first)
outlines:        Produces LFS300_{short,long,appendix}_outline.tml (do make first) and pdf outlines
checklinks:     Checks all URLs and file and kernel source links
                can also do just checkurls, checkefilelinkes, checkkernlinks
SOLUTIONS       Produces the SOlUTIONS directory
COVERS          Produces full, front and slides covers
RELEASE/release Populate RELEASE directory with all PDFS with version numbers embedded
release-elearning Populate RELEASE directory with all PDFS with version numbers embedded for ELEARNING
↪   COURSE
generic         Make a non branded release with no course number
release-generic Same as generic, but no SOLUTIONS or RESOURCE files
resources-upload   sync up to   ../RESOURCES/LFS300
resources-download sync down from ../RESOURCES/LFS300
SOLUTIONS-upload   SOLUTIONS tar ball To cm website
watermark/WATERMARK Produce watermaked copies
labs               Produce a labs only version
labsplit/splitlabs Produces separate files for each lab+solution
spellcheck      Check spelling
```

```
spellcheck-list-files  List files that would be spell-checked
FINDOVERFULL    Find vertical and horizonal overruns
FINDLONG        Find too long lines in raw sections
help HELP or usage: Produces this message
```

Not all targets are useful for all authors. In particular, the targets involving `resources` and `upload` cannot be done by non-privileged users.

A helpful hint when working on exercises

When you are working on exercises the command:

On the command line

```
$ make labs
```

which produces `LFS300-labs.pdf` may be useful as it produces a file with only the labs, and so it builds faster.

6 Structure and Style

There are some aspects to how the courseware should be structured and styled:

- To date we avoided having more than one level in each session; i.e., each session has subsections, but not subsubsections, and we advise splitting sessions up to make things less bulky. LaTeX does permit subsubsections (not surprisingly through the `\subsubsection{}` command) but we would have to tweak the table of contents depth options to show them.
- Most sessions (if not all) should have laboratory exercises, with the last slide in the session being `lab.tex` (or `labs.tex`, both names are fine.) Solutions should be placed in a `labs` subdirectory under the session directory.
- Almost everything should be in bullet point format, itemized and enumerated lists. This is not absolute, it's just style. You can use the `\subsection*` tags to set off subsections within a section; the asterisk ensures they won't get picked up in the table of contents, which is fine.
- Commands should be in bold, like **bash**, rendered by `\textbf{bash}`. To avoid interpretation as LaTeX special characters you can use various **verbatim** commands and environments as discussed later; the simplest way would be something like `\verb?ls -l file_with_underscore?`.
- Filenames, paths etc, should look like like `/etc/passwd`, which is rendered by the macro `\filelink{/etc/passwd}`, or as `passwd` which is rendered by `\file{passwd}` when it is not a filename that can actually be linked to.
- Remember that the same source input will be used for both the classroom projection material and the printed book, so when you do links, put in the full link, rather than saying something like "click **here**".
- Be careful with quotation marks to get left and right correct. If you are working in a smart editor (like **emacs**) it will do it for you automatically. However, the correct LaTeX way to do it is: ``` ``quoted content'' ``` which will print as "quoted content".

The above rules are not rigid, and we don't mean to inhibit creativity. But the more you follow it the more our courseware will look uniform, and the less work I'll have to do in editing and production.

7 Some General Considerations

Here are some style and procedural considerations:

- Avoid very long lines inside **verbatim** environments. Depending on context this means 90-100 characters. (Inspect the pdf for lines running off the page.)

  There are some advanced tricks to use smaller fonts to avoid problems, but often they have to be broken up by hand. The breakup can sometimes alter intent and it is best if the author does it, rather than the "copyeditor". Most of the time verbatim sections should not be indented much or at all and that helps.

- Please don't overdo **boldface**, *italics* and CAPITALS. It gets very tiresome and one loses energy trying to decide where it is really justified (which is almost never.) It insults the audience (it's like flaming in all caps in email) saying **"Listen you *MORONS*, this is important but you are unlikely to see it unless I drive it through your puny little brains!"**

- Remember we will be converting all figures to grayscale before printing. It's great to use color figures when teaching, but occasionally things become invisible when they are converted.

The most time consuming thing can be avoiding **line overruns** and **bad page breaks**, but improvements we have made to the style files have helped a lot with this.

`\linebreak` vs `\newline`

Note that `\linebreak` forces a new line and spreads text out to a justified margin. If you do `\newline`, or a simple `\\`, or just insert a blank line, it simply truncates the line without any justification. Which you use depends on what looks best.

You can force a new page with: `\newpage`. This is not at all the same as `\clearpage` which may or may not give you a new page; it should be called "clear floats".

The worst problems with overruns come from long hyperlinks, even though we have made use of a bunch of LaTeX tricks to tell it when it can split up a URL across lines. Sometimes the only thing one can do is to manually manipulate the link to avoid this. This has the side effect of screwing up the link in the projection material. With luck you will not have these problems and your copyeditor will look at these.

The page break thing is no big deal, but it has to be manual. We try to use as few of these as possible and every time there is a new edition we often wipe out the old ones and figure it out all over again. Besides the "widows and orphans" problem it involves trying to make sure the figures appear in the text where they are spoken of. Sometimes this is a royal pain and that's why you will see some pages with a lot of white space. It is better to have some white space than a misplaced figure.

This is where the `\checkclearpage{}/` statements come from. Each one tells **LaTex** to clear out the *floats* and do a page break. The reason the class name is embedded in the command is sometimes we use the same files in more than one class and the page breaks will come out differently.

The procedure you can use is to do:

1. Run `make` and then `make findoverfull` :

On the command line

```
$ make
    ....
$ make findoverfull
    FINDING VERTICAL OVERRUNS .........
FINDING HORIZONTAL OVERRUNS .........
LFS300.log-File: IMAGES/vimubuntu.png Graphic file (type png)
LFS300.log-<use IMAGES/vimubuntu.png>
```

```
LFS300.log-Package pdftex.def Info: IMAGES/vimubuntu.png  used on input line 15.
LFS300.log-(pdftex.def)             Requested size: 241.26569pt x 144.68323pt.
LFS300.log-
LFS300.log:Overfull \hbox (21.93445pt too wide) in paragraph at lines 15--16
LFS300.log- [][]
LFS300.log- []
LFS300.log-
--
LFS300.log-File: IMAGES/emacsubuntu.png Graphic file (type png)
LFS300.log-<use IMAGES/emacsubuntu.png>
LFS300.log-Package pdftex.def Info: IMAGES/emacsubuntu.png  used on input line 16.
LFS300.log-(pdftex.def)             Requested size: 241.26569pt x 211.90814pt.
LFS300.log-
LFS300.log:Overfull \hbox (21.93445pt too wide) in paragraph at lines 16--17
LFS300.log- [][]
LFS300.log- []
LFS300.log-
...
```

By looking at the output you can see what text in what file caused the overfull hboxes. By default, we do not accept anything more than about 10 points as they cause bleed into margins.

2. Find each problem, put in line breaks and edit text to avoid overfull hboxes
3. Deal with overfull vboxess. You will see them in the text as either text hitting the bottom of the page or horizontal lines at the bottom of pages. Fix with appropriate use of `\clearthepage{LFD000}` and other editing.
4. Iterate until done.

Very Important

Note this often does **not** catch overrun in verbatim environments, so you must actually eyeball the pdf. No way around it.

You also want to avoid “widows and orphans”. In other words avoid having sections start at the bottom of the page of having only small spill over of a section to the next page. Once again it is cosmetic and manual to avoid.

In practice the worst things are insane hyperlinks that are like 200 characters (some of which are on our own web pages). There is no way to chop them up properly and have them work as links (unless you want to figure out the `\slash` LaTeX macro, but I've stopped doing this as too laborious.)

8 `common` Directory Files

The directory `LFS300/common` (linked to `../common`) contains a bunch of files used by all classes, some of which are symlinked:

1. `Makefile_oneclass` is either linked to or copied to `Makefile` for each class, according to whether or not you have only one class or not
2. The `texmf` directory tree contains all the LaTeX style files etc as well as some graphics used in covers etc.

On the command line

```
c7:/tmp/LFCW>ls -l common/texmf/tex/latex/LFCW
total 320
-rw-rw-r-- 1 coop coop   1904 Dec  9 07:19 japanesetitles.tex
-rw-rw-r-- 1 coop coop  17076 Apr  1 09:59 lf-boxes.tex
-rw-rw-r-- 1 coop coop   4516 Feb 24 10:25 LFcover.cls
-rw-rw-r-- 1 coop coop  16231 Apr  1 09:59 LFD.cls
-rw-rw-r-- 1 coop coop  24254 Apr  1 09:59 lf-listings.tex
-rw-rw-r-- 3 coop coop   5873 May  7  2019 linuxfoundation-cropped.png
-rw-rw-r-- 2 coop coop  18289 May  7  2019 linuxfoundation.pdf
-rw-rw-r-- 2 coop coop 207252 May  7  2019 logo-training.pdf
-rw-rw-r-- 1 coop coop   1810 Sep 19  2019 longcopyrightjapanese.tex
-rw-rw-r-- 2 coop coop   1030 Apr 23  2019 longcopyrightspanish.tex
-rw-rw-r-- 1 coop coop   1536 Dec  9 07:19 longcopyright.tex
-rw-rw-r-- 1 coop coop   1544 Sep 19  2019 spanishtitles.tex
```

We will not discuss the remaining contents of the **common** repo as course authors will not need to use them or edit them directly, but you are welcome to study them to see how the **make** targets function.

9 Other make Targets

The `Makefile` in the main directory is the same for all courses.

We already talked about these targets:

On the command line

```
$ make
  $ make slides
  $ make labs
  $ make splitlabs
```

Normally you only need to run the first two. A few others you might need to consider or find useful:

- `outlines`

  Produces short and long (detailed) outlines in both `html` and `pdf` form. These are used only for the website and other propaganda and sales purposes.

- `checklinks`

  Extracts the URL's from the source and checks their validity, giving output like:

```
GOT 200: Good Link at:      http://www.gnu.org/software/global
GOT 200: Good Link at:      http://www.kernel.org
GOT 401: Unauthorized:      http://training.linuxfoundation.org/cm/course
GOT 200: Good Link at:      http://www.kroah.com/lkn/
GOT 200: Good Link at:      http://www.kernel.org
```

- `SOLUTIONS`

  This fills in the content in the `SOLUTIONS` directory, which should already exist. We'll have a separate section on this later. This also populates the `RESOURCES` directory if the course has that one as well. (Note that you don't have to do a make before doing `make SOLUTIONS`.)

- `COVERS`

  Produces 3 covers; full front/back, front only, front only for slides.

- `help`

  Prints out all possible targets; remember default is `class`.

- `clean`

  Wipe out everything that can be recreated, leaving input files alone.

Please Note

Note, if you examine the `Makefile`, you will notice that multiple invocations of **pdflatex** and **makeindex** are required ; this is necessary to build properly the table of contents and the index.

10 Master LATEX file

An example master LATEX file would look like:

LFD420.tex

```
\newcommand{\NOSUDOLAB}{}  %% Just to exclude lab in intro chapter

\newif\ifINSTRUCTORLED
%% For Instructor Led:
\INSTRUCTORLEDtrue

\newcommand{\course}{LFS300}
\newcommand{\arch}{x86}
\newcommand{\version}{4.1}
\newcommand{\kversion}{\version}
\newcommand{\coursetitle}{Fundamentals of Linux}

\documentclass{LFD}

\renewcommand{\includefootnote}{}
%%\renewcommand{\includelistoftables}{}
%%\renewcommand{\includelistoffigures}{}

\begin{document}

\topshit

\subimport{CHAPS/introduction/}{index}
\subimport{CHAPS/prelim/}{index}
\subimport{CHAPS/linux-concepts/}{index}
\subimport{CHAPS/boot/}{index}
\subimport{CHAPS/install/}{index}
\subimport{CHAPS/graphical-interface/}{index}
\subimport{CHAPS/graphical-sys-config/}{index}
\subimport{CHAPS/documentation/}{index}
\subimport{CHAPS/common-apps/}{index}
\subimport{CHAPS/editors/}{index}
\subimport{CHAPS/command-line/}{index}
\subimport{CHAPS/user-environment/}{index}
\subimport{CHAPS/text-manipulation/}{index}
\subimport{CHAPS/file-operations/}{index}
```

```latex
\subimport{CHAPS/bash-scripting/}{index}
\subimport{CHAPS/processes/}{index}
\subimport{CHAPS/printing/}{index}
\subimport{CHAPS/network-operations/}{index}
\subimport{CHAPS/local-security/}{index}
\subimport{CHAPS/closing/}{index}
\end{document}
```

The environmental variables set above should be clear enough. (`arch` must be set, usually to **x86**, but for some classes we set it to `ARM`.)

If you want to include optional sections (which will be denoted with ** in the chapters) the comment:

Please Note

** These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

will be put in the table of contents. If you are not doing this remove the %% from the line

```latex
%%\renewcommand{\includefootnote}{}
```

to uncomment it and not insert the text. Likewise, uncommenting either of the lines

```latex
%%\renewcommand{\includelistoftables}{}
%%\renewcommand{\includelistoffigures}{}
```

will prevent having a list of tables and/or figures in the table of contents, if you are not using them.

The session or chapters should each be a separate directory under `CHAPS` containing an `index.tex` file, to be included with the `\subimport` macro as shown. Each directory should be prefixed with the name of the course, as in `LFD000_`. This actually doesn't matter for producing the printed output but at present matters for packaging up the solutions.

Finally the `\printindex` command prints the index. If you are not indexing comment out this line.

Custom Macros

If you want to define custom macros (commands in LaTeX language) this is the best file to do it in. For example:

```latex
\newcommand{\yoctoversion}{\version}
\newcommand{\yoctobase}{1.5.1}
\newcommand{\pokyversion}{10.0.1}
\newcommand{\pokybase}{10.0}
\newcommand{\pokyname}{dora}
\newcommand{\yoctokernel}{3.10}
% bbone
\newcommand{\arch}{arm}
\newcommand{\bboneboard}{}
```

```
%comment out for minnowboard
%\newcommand{\arch}{x86}
%\newcommand{\minnowboard}{}
%
```

where you notice anything after a % sign is a comment.

Conditionals

Note: LaTeX also has conditional expressions, but it is little complicated to get into here, but you can see it used in the **LFD.cls** file.

11 Session Index

Each section directory must contain a **index.tex** file. Here's an example from an **LFD** course:

LFD Style: index.tex

```
\chapter{Monitoring and Debugging}
\lflogo

We consider various techniques used to debug kernel code.
We consider how to trace system calls and do profiling and
testing.  We discuss using the \textbf{SysRq} key.  We look
at dissecting \textbf{oops} messages, and the use of
debuggers.  We also consider the use of \textbf{debugfs}.

\minitoc

\optionalfootnotetext
\section{Debuginfo Packages }
\input{debuginfo}

\section{Tracing and Profiling}
\input{mon}

\section{\textbf{sysctl}}
\input{sysctl}

\section{SysRq Key}
\input{sysrq}

\section{\textbf{oops} Messages}
\input{oops}

\section{Kernel Debuggers}
\input{debug}

\section{\textbf{debugfs} }
\input{debugfs}

\section{Labs}
\input{lab}
```

and an example from an **LFS** couurse:

LFS STYLE: index.tex

```
\chapter{Printing}

\lflogo

\minitoc

\clearpage

\input{config}
\input{operations}
\input{pspdf}
\input{labs}
```

Note:

- The **LFD** style always has a short description of the learning objectives for the section. There is no reason (other than historical) not to do this in **LFS** style, other than it can not be too long if it is going to fit in the slide frame.
- `\minitoc` produces the miniature table of contents at the beginning of the chapter.
- The `\section{}` lines can go in the `.tex` file being included or in `index.tex`. Note it is not necessary to give the `.tex` extension. There can also be multiple sections in one file.
- Only include `\optionalfootnotetex` if there are optional sections denoted by **.

12 Frames, Slides, and Notes

Very Important

This section is not for LFD courses This section is only relevant to **LFS *slides + notes style*** courses.

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

Very Important

- `LFS300-SLIDES.pdf` contains only the slides and is what the instructor projects.
- `LFS300.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 (which we have completely redefined) etc.: Beamer is extremely limited in its capabilities and uses only a small subset of LaTeX.

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:

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

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

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

temporary.tex

```latex
\section{Creating Temporary Files and Directories}

\begin{frame}
   {Temporary Files and Directories}

   \begin{itemize}
      \item Scripts often need to create or use temporary files
      \begin{itemize}
         \item Typically used to store data not used after the
         task is accomplished
      \end{itemize}
      \item Creating a \textbf{random} filename is the best
      practice
      \begin{itemize}
         \item Only accessible to user who creates the
         file (\verb?600?)
      \end{itemize}
      \item Use \textbf{mktemp} with \verb?XXXXXX? to define the
      random field:
      \begin{cmd}
$ mktemp /tmp/temp.XXXXXX
      \end{cmd}
      \item Creating temporary directories can be done with the
      \verb?-d? option:
      \begin{cmd}
$ TEMP=$(mktemp -d /tmp/tempdir.XXXXXXXX)
$ mv file $TEMP
      \end{cmd}

   \end{itemize}

\end{frame}

\cprotect\note{

   Just using a simple file name to create temporary files
   can open your script to attacks. If someone with access
   to the \filelink{/tmp} directory reads that your script
   does something like this:

   \begin{cmd}
$ echo $VAR > /tmp/tempfile
   \end{cmd}

   The attacker can then do something like:
```

```
   \begin{cmd}
$ ln -s /etc/passwd /tmp/tempfile
   \end{cmd}

   When the script will be executed by the root user, the
   password file will thus be overwritten. In order to
   avoid this situation make sure you randomize your
   temporary filenames by replacing the line above with:

   \begin{cmd}
$ TEMP=$(mktemp /tmp/tempfile.XXXXXXXX)
$ echo $VAR > \TEMP
   \end{cmd}
}
```

You must have a note section in each frame

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. Both should have **captions**. We will discuss later.

Notes are only in the book

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.

13 Covers

The **make COVERS** target produces three auto-generated outputs:

1. `LFS300-COVER-FRONT.pdf`:
   Full cover front cover, also used for first interior page.
2. `LFS300-COVER-BACK.pdf`:
   Full cover back cover; actually the same for all classes and versions.
3. `LFS300-COVER-SLIDE.pdf`:
   Full cover slide cover, used for first page of slides.

These are built using the classes defined in the `texmf` sources, `LFcover.cls`.

Because the front cover is included as the first page in the book the cover must be built before the book can be built. But it need be done only once unless version numbers change.

14 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.

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 and commands and output can be done with a variety of different methods. First we will discuss several which are not sensitive to their contents (i.e., is it **C** or **Python** code, a configuration file, etc.) but which we have created for specific purposes.

Commands and Output: `cmd` and `response`

We often get a situation where we want to display a command and then show the output that appears on the terminal, as in:

```
$ ls -l
```

```
total 4
-rw-rw-r--. 1 rkimble rkimble    0 Mar 16 19:04 file-1
-rw-rw-r--. 1 rkimble rkimble    0 Mar 16 19:04 file-2
```

The LaTeX that produces this is:

```
   \begin{cmd}
$ ls -l
   \end{cmd}
   \begin{response}
total 4
-rw-rw-r--. 1 rkimble rkimble    0 Mar 16 19:04 file-1
-rw-rw-r--. 1 rkimble rkimble    0 Mar 16 19:04 file-2
   \end{response}
```

(By the way, the reason we use `response` instead of `output`, is that the latter is a reserved word in LaTeX.)

Note the following:

- Different colors are used for the command typed (blue) and the response (purple).
- The font is smaller than the normal text font, which is a good thing to do for monospace fonts.

Multiple verbatim lines: `kcode`

A similar macro is provided by

```
rwxrwxrwx  rw-rw-r--  rwxr-xr-x  r--r-----
 7  7  7    6  6  4    7  5  5    4  4  0
```

```
     \begin{kcode}
rwxrwxrwx  rw-rw-r--  rwxr-xr-x  r--r-----
 7  7  7    6  6  4    7  5  5    4  4  0
     \end{kcode}
```

which displays in the color sepia. This is good for multiple lines of verbatim text, perhaps file content, or an email message etc.

verbatim with embedded LaTeX: `cmdtt` and `responsett`

Sometimes you need to embed macros inside of verbatim sections. For example, if in the main master tex file we have

```
\newcommand{\course}{LFS300}
```

the following lines:

```
     \begin{cmdtt}
$ echo Welcome to \course
     \end{cmdtt}
     \begin{responsett}
Welcome to \course
     \end{responsett}
```

will produce:

```
echo Welcome to LFS300
```

```
Welcome to LFS300
```

A somewhat more complicated example would be:

```
$  unxz $RESOURCES/s_\thesessionref/\targetimagexz
```

Notice you have to properly escape the special character in the above (_) as _), as otherwise LaTeX will fail to compile to a pdf without it.

These macros can be a little tricky to use, so feel free to ask for help if things come out looking funny.

Including entire files verbatim

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

LATEX
```
\VerbatimInput{examples/file.sh}
\verbatiminput{examples/file.c}
```

However, we prefer you use our custom macros instead, which are smarter.

If you want to put in long output without putting directly in the .tex file you can simply use `\respfile{}` as in:

LATEX
```
\respfile{ldd.output}
```

which will insert `ldd.output`.

We will detail later a number of macros that are customized for certain kinds of files. For example

LATEX
```
\shfile[file.sh]{examples/file.sh}
```

produces

SH **file.sh**
```
echo $PATH > doo.txt
ls -l
exit 0
```

(The title `[file.sh]` is optional). You can also produce the identical output by doing:

LATEX
```
     \begin{shlst}[file.sh]
echo $PATH > doo.txt
ls -l
exit 0
       \end{shlst}
```

There are many other such environments and file inclusion macros, to be detailed later, including: `\cfile{}`, `\fromfile{}`, `\frommakefile{}`, `\shfile{}`, `\gitfile()`, `\configile{}`, etc.

15 Hyperlinks

The simplest way to do a hyperlink is with:

`\url{`https://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}`.

Links Without Extensions

Note this is likely to fail on files without extensions and results will vary according to what pdf browser you use. Customizing exactly what application opens when you hit a link can be enough to make you pull your hair out and we will not discuss it here.

Both `\url{}` and `\filelink{}` can properly handle references with special characters, such as _.

For links to the **Linux kernel source** use:

`\kernlink{kernel/signal.c}`.

This will produce `kernel/signal.c` in the visible text, but link back to `/usr/src/linux/kernel/signal.c`.

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

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

You can always do this if necessary, but we don't recommend having the link being different than the highlighted text as it will not appear in the book properly.

Long hyperlinks

Very long hyperlinks can present problems as they tend to run off the page. Sometimes using `\href{}` works better than `\url{}` because of font differences. One can also change the fontsize as in:

```latex
\small{\url{https://training.linuxfoundation.org}}
```

Finally, there is a special `\lablink{}` macro, to be used only in `labs.tex` files. This link inserts the string `SOLUTIONS/s_\thesection/` in front of the file name. This is very important.

If you untar the `SOLUTIONS` file in the same directory you are displaying the pdf file from, you can click on say `lab1.sh`, and the link will resolve to `SOLUTIONS/s_n/lab1.sh` where `n` is the session number. This permits you to move sections around and not have to renumber anything! Quite cool if I say so myself.

16 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:

```latex
\begin{figure}[H]
  \includegraphics[scale=0.60]{IMAGES/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.

Figure Placement

The [H] in the figure code above says put this figure here, do not float it elsewhere to save white space. For example one often uses [hbt] to gain more flexibility. We do not recommend this as you have to go hunting for figures when you are reading the text.

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 width=0.8\textwidth) or [height=xx.in] etc.; you will have to play with values for appearance. For slides which contain only images, usually height=3.3in or width=6.0 in may work nicely

It is possible to put the images in other locations besides the IMAGES directory, but we have found that to be irritating in some cases.

17 Tables

LaTeX has a number of methods of making tables, including longtable and tabular, and they will work just fine in our courseware.

However, we prefer the following method. As an example:

Table 17.1: **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

which is rendered by:

LaTeX

```
\begin{table}[H]
   \caption{printk() Log Levels}
   \vspace{-15pt}
\end{table}

\small{\begin{tcolorbox}[tabularx={X|p{0.10\textwidth}|p{0.70\textwidth}}]
      \textbf{macro}       &  \textbf{Level}  &  \textbf{Meaning}   \\ \hline
      \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{tcolorbox}}
```

The caption will appear in the list of tables and is important.

In the above example we have manually determined the width of the columns. It is possible to have this be automatically configured, by replacing the explicit lengths by just X, as in

Table 17.2: **Page Table Levels**

Arch	**Page Size**	**Paging Levels**	**Address Bits**	**Offset Bits**	**Bits in Levels**
alpha	8 KB	3	43	13	10 + 10 + 10
ia64	4 KB	3	39	12	9 + 9 + 9
ppc64	4 KB	3	41	12	10 + 10 + 9
x86_64	4 KB	4	48	12	9 + 9 + 9 + 9

rendered by

```
\begin{table}[H]
   \caption{Page Table Levels}
   \vspace{-15pt}
\end{table}

\small{\begin{tcolorbox}[tabularx={X|X|X|X|X|X}]
      \textbf{Arch} & \textbf{Page Size} & \textbf{Paging Levels} & \textbf{Address Bits} &
      ↪  \textbf{Offset Bits} & \textbf{Bits in Levels} \\ \hline\hline
      alpha   & 8 KB & 3 & 43 & 13 & 10 + 10 + 10     \\ \hline
      ia64    & 4 KB & 3 & 39 & 12 &  9 +  9 +  9     \\ \hline
      ppc64   & 4 KB & 3 & 41 & 12 & 10 + 10 +  9     \\ \hline
      x86\_64 & 4 KB & 4 & 48 & 12 &  9 +  9 +  9 + 9
   \end{tcolorbox}}
```

It is nice to just get automatic widths, and it is often the best way to start, but sometimes entries overrun column widths, and sometimes they just look bad.

It is often a good idea to split (longer) tables off into separately included files, but this is just a way of working that is not necessary.

18 LABS

Each section should have a `labs.tex` (or `lab.tex`) file. It may also have a `labs` subdirectory as will be explained in the next section on SOLUTIONS.

Here is a simple example:

Exercise 18.1: Examining socket options

- Write a program that uses `getsockopt()` to examine the default options for a **TCP/IP** socket.
- Just open a socket (you do not have to do anything with it, like connecting), and then using the option names available in `include/uapi/asm-generic/socket.h` or `arch/x86/include/uapi/asm/socket.h` in the kernel source, obtain their values.
- Most of the options have an integer value, but some are not, such as those which are given in a `timeval` data structure.
- Print out the symbolic name along with the integer option number.
- For the above you will use `SOL_SOCKET` for the `level` parameter. You can also try other values such as `SOL_IP`. Look at the **man** pages for **socket(7), tcp(7), ip(7), unix(7)**, etc. to get some ideas.

Solution 18.1

lab_options.c

which is rendered by:

Example Lab

```
\begin{Lab}
   \begin{exe} {Examining socket options}
      \begin{itemize}
         \item

         Write a program that uses \verb?getsockopt()? to
         examine the default options for a \textbf{TCP/IP} socket.

         \item

         Just open a socket (you do not have to do anything with
         it, like connecting), and then using the option names
         available in
         \kernlink{include/uapi/asm-generic/socket.h} or
         \kernlink{arch/x86/include/uapi/asm/socket.h} in the kernel source,
         obtain their values.

         \item

         Most of the options have an integer value, but
         some are not, such as those which are given in a
         \verb?timeval? data structure.

         \item

         Print out the symbolic name along with the integer option
         number.

         \item

         For the above you will use \verb?SOL_SOCKET? for the
         \verb?level? parameter. You can also try other values
         such as \verb?SOL_IP?. Look at the \textbf{man} pages for
         \textbf{socket(7), tcp(7), ip(7), unix(7)}, etc. to get
         some ideas.

      \end{itemize}
      \begin{sol}
         \begin{alltt}
   \lablink{lab_options.c}
         \end{alltt}
      \end{sol}
   \end{exe}
\end{Lab}
```

Multiple exercises in a chapter

You can have as many exercises as you need, as long as they each have a `\begin{exe}` and `\end{exe}`, as in

```
\begin{Lab}
  \begin{exe} {First lab title}
    ....
  \end{exe}
  \begin{exe} {Second lab title}
    ....
    \begin{sol}
      ....
    \end{sol}
  \end{exe}
\end{lab}
```

Please Note

- 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}`.)
- If the section does not have labs, labs.tex could include something simple like:

Empty lab

```
\begin{Lab}
  \begin{exe} Lab
    There is no lab to complete for this chapter.
  \end{exe}
\end{Lab}
```

or you can decide do just not have a lab file.

19 SOLUTIONS and RESOURCES Archives

When students take the course, they will be able to download any existing `SOLUTIONS` and `RESOURCES` tarballs (archives) from the course web site, at https://training.linuxfoundation.org/cm/LFS??? (where you need to put in the proper course number.) These files have names which incorporate course number, name, and version, as in:

On the command line

```
$ ls -l   *SOLUTIONS* *RESOURCES*
-rw-rw-r-- 2 coop coop 230809600 Apr  6 14:27 LFS300_V4.1_RESOURCES.tar
-rw-rw-r-- 2 coop coop      3870 Apr  6 14:27 LFS300_V4.1_SOLUTIONS.tar.bz2
```

Not all courses have `SOLUTIONS` and even fewer have `RESOURCES` archives.

SOLUTIONS

For SOLUTIONS one starts with a skeleton directory in the source. For a course that contains code, you would have:

On the command line

```
$ ls -l SOLUTIONS
-rwxrwxr-x 1 coop coop  4011 Feb 13 09:04 genmake*
-rw-rw-r-- 1 coop coop 15251 Feb  8 16:08 LICENSE
-rw-rw-r-- 1 coop coop   991 Feb 13 09:03 Makefile
```

LICENSE contains information about how the code is licensed (under **GPL v2**) and copyrighted. The Makefile, together with genmake are used to compile all the code in the subdirectories that will be created.

Whether or not you have source code, shell scripts, **yaml** files etc, when you do:

On the command line

```
$ make SOLUTIONS
```

the subdirectories s_01,s_02,...s_xx are created and populated with any files from the labs directories in the sessions in CHAPS subdirectories. Numbering is automatic.

In addition any files with an extension of **.c, .h, .sh** are copied over to the EXAMPLES subdirectory under the s_* directories. Any empty directories are wiped out.

Furthermore all **.c, .h** files have a notice put at their beginning that looks like: following:

```
/* **************** LFD420:5.6 s_20/lab_wastemem.c **************** */
/*
 * The code herein is: Copyright the Linux Foundation, 2020
 *
 * This Copyright is retained for the purpose of protecting free
 * redistribution of source.
 *
 *     URL:    https://training.linuxfoundation.org
 *     email:  info@linuxfoundation.org
 *
 * This code is distributed under Version 2 of the GNU General Public
 * License, which you should have received with the source.
 *
 */
/* simple program to defragment memory, J. Cooperstein 2/04
 */
```

Note the first line is autogenerated and includes the course name, the version, and the file name. The rest is boilerplate.

Shell files (**.sh**) receive the same except everything is prefixed with #.

Please Note

If you are referring to certain files in the solutions in the text (almost always in lab descriptions) it should be done as:

```
Cut and paste from
    \filelink{/home/student/LFT/\course/SOLUTIONS/s_\thesessionref/blah.sh}
```

Kernel modules have the following macros at their end:

```c
MODULE_AUTHOR("Jerry Cooperstein");
MODULE_DESCRIPTION("MODULE_DESCRIPTION_NAME");
MODULE_LICENSE("GPL v2");
```

(Of course use the right name). The macro with the string MODULE_DESCRIPTION_NAME will be converted to something like:

```c
MODULE_DESCRIPTION("LFD000:1.0 s_03/lab1_kprobes.c");
```

by the **make SOLUTIONS**. Please use this.

RESOURCES

The RESOURCES archive is intended as a container for relatively large files, usually binary source archives, jar files etc. These can be quite large: we have developer courses where these files can be several GB.

git is not really designed to handle large or binary files and they do not belong in the repository for the course. Instead we maintain them in a directory on the **Linux Foundation** server as well as on our local machines.

If you need to take advantage of the RESOURCES infrastructure you will need to have a directory tree on your machine to hold it, that you can link back into from your CHAPS/*/labs directories.

RESOURCES contains everything that is in the labs directory other than basic code files and scripts.

Very Important

The way the build system knows these are files intended for RESOURCES and not SOLUTIONS is rather simple: any file that is symlink'ed goes to RESOURCES.

There is a subdirectory such as RESOURCES/LFD000 under which anything linked to in MODULES/LFD000_*/labs/ points to. So you have things like:

On the command line

```
$ ls -lF LFS300/CHAPS/linux-concepts/labs
total 0
lrwxrwxrwx 1 coop coop 55 Apr  9 12:37 using_linux_distros_demo.mp4 ->
↪  /teaching/RESOURCES/LFS300/using_linux_distros_demo.mp4
```

There are targets in our build Makefile for uploading and downloading RESOURCES files, but only **Linux Foundation** personnel have the requisite access rights, at this time, but we are hoping to make this more easily accessible to other course authors.

20 Custom Commands and Environments

The file `../common/texmf/tex/latex/LFCW/LFD.cls` includes all necessary packages and defines custom commands and environments used by LaTeX for every class. Some are just internal and you will not use them. It also includes `../common/texmf/tex/latex/LFCW/lf-boxes.tex` and `../common/texmf/tex/latex/LFCW/lf-listings.tex` which have an assortment of environments and ways to show various kinds of content.

Here is a short list of custom **commands** you might use:

- `\thesessionref`
  Extracts the session number with two digits (i.e., `s_03` not `s_3`). This is generally only used in laboratories to point to files in the solutions or resources properly.
- `\filelink{}`
  Used for links to local system files (we use `\url{}` for network hyuperlinks.
- `\file{}`
  Like `\file{}` except you cannot actual click on a link. Used when you want to just say something is a file and have the name colored
- `\kernlink{}`
  Used for links to the **Linux** kernel source
- `\lablink{}`
  Used for links to `SOLUTIONS` in labs. If the instructor shows the class slides in the same directory as `SOLUTIONS` is expanded these links will show the file content.
- `\reslink{}`
  Used for links to `RESOURCES` in labs. If the instructor shows the class slides in the same directory as `RESOURCES` is expanded these links will show the file content.
- `\optionalfootnotetext`
  Used in section **index.tex** files when there is optional material.
- `\checkclearpage{LFDxxx}`
  This produces a page clear specific to a class. For example

  ```
  \checkclearthepage{LFD420}
  ```

  will produce `\clearpage` **only** if the class is **LFD420**. (This is useful in text that might be used in more than one class, please do avoid using just `\clearpage`.)

Please Note

`\clearpage` does not induce necessarily a **page break**; it does force all pending figure, table inclusions etc to flush out. It is a manual process to get these right.

Pages not cleared with slide deck, only in book form

The `\checkclearpage{}` macro is ignored while producing the slide deck as the breaks would be in appropriate. There is a `\clearpageslides` which just does a `\clearpage` but I've never bothered to use it.

- `\SOURCEFILE{}`
  This just inserts a special comment indicating the (kernel) source file that the text was extracted from. LaTeX ignores the argument (as in `\SOURCEFILE{/usr/src/linux/include/kernel.h}`) but we have scripts which use this information to properly introduce annotated source in the material.

21 Custom Boxes

21.1 Custom Information Boxes

There are a number of custom informational **box** environments you might use to catch the eye of the user so they will see what sort of thing the box is about at a glance. You can change their titles, and they all handle page breaks appropriately.

These boxes work across page breaks.

The simplest is the **lfbox**:

```
\begin{lfbox}
  You can put whatever you want in here.

  Including nested boxes of any type.
\end{lfbox}
```

which produces:

You can put whatever you want in here.

Including nested boxes of any type.

You can also add a title as in:

```
\begin{lfbox}[This is an Example Title]
  You can put whatever you want in here.

  Including nested boxes of any type.
\end{lfbox}
```

which produces:

This is an Example Title

You can put whatever you want in here.

Including nested boxes of any type.

There are a number of other informational boxes, which are all done the same way. The others all include an icon image and have a default title.

We will show the code only for the `info` box as they all follow the same exact method.

- `info`
  As an example:

```
\begin{info}
      This is something you might find interesting beyond
      what was said before.
    \end{info}
```

renders

Please Note

This is something you might find interesting beyond what was said before.

If you want to change the title just add it like this:

```
\begin{info}[A Custom Title for the info BOX]
  This is something you might find interesting beyond what
  was said before.
\end{info}
```

rendering:

A Custom Title for the info BOX

This is something you might find interesting beyond what was said before.

- `important`

Very Important

You really need to read this and heed its advice!

- `howitworks`

How it works

This is to provide advanced information on the topic at hand, like deeper details on how things work.

- `kvnote`

Kernel Version Note

Used to create a box with a cute penguin to include linux kernel version dependencies:

- `dontdothis`

Don't do this

This is to catch the eyes of students and tell them **Not to to this!** or **really bad things will happen**™

- `question`

Question

This is to indicate a question.

21.2 Custom Highlight Boxes

Here is a list of custom **boxes** you might use to highlight specific things. You can change their titles, and they all handle page breaks appropriately. The coding works exactly the same as for the Information Boxes.

- `configbox`
  To label changes that need to be made to a configuration file.

make menuconfig

In the ”General setup” menu:

- Enable ”**Kernel .config support**” (CONFIG_IKCONFIG)
- Enable ”**Enable access to .config through /proc/config.gz**” (CONFIG_IKCONFIG_PROC)

Note the default title is **make menuconfig**.

- `gitbox`
  This box is intended for providing the git alternative to installing otherwise from tarball in class. You may need to use a `cmdtt` environment inside this to allow you to use LATEX macros in shell code. You can also just use it to give a sequence of **git** commands.

You could also get this upstream from git

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

```
Cloning into 'linux-stable'...
remote: Counting objects: 7177449, done.
remote: Compressing objects: 100% (1096739/1096739), done.
remote: Total 7177449 (delta 6036208), reused 7174727 (delta 6034078)
Receiving objects: 100% (7177449/7177449), 1.28 GiB | 10.66 MiB/s, done.
Resolving deltas: 100% (6036208/6036208), done.
Checking out files: 100% (62470/62470), done.
```

- `videobox`
  This box is intended for showing off example movies used as optional homework resources. This is only done in instructor-led courses as e-learning courses have videos embedded

Video Demonstration Resources

example/video.mp4
example/video.mp4

- `webbox`
  This box is intended for highlighting content available on the web.

On the Web

You can find out more by going to: http://training.linuxfoundation.org

21.3 Linux Distribution Boxes

While we try to keep **Linux Foundation** courses as distribution-flexible as we can, and we work hard to ensure they work on all the major Enterprise versions. However, there will always be things that have to be done differently due to actual system being used for the course (which will often differ among students even in the same session.)

Thus we have a list of custom **Distribution Boxes** which encapsulate distribution-dependent material. As an example:

```
  \begin{ubuntubox}
    \begin{cmdtt}
$ cat /etc/debian_version
$ systemctl status libvirtd
    \end{cmdtt}
  \end{ubuntubox}
```

renders as

On Ubuntu

```
$ cat /etc/debian_version
$ systemctl status libvirtd
```

Here are examples of the of the full list of current choices. They are all rendered in the same manner. If you want to use a custom title other than the default ones here, just put it in `[..]` at the beginning as we did for informational and highlight boxes.

- `archlinuxbox`. `\archlinux` renders as **Arch Linux**

  On Arch Linux

  ```
  $ cat /etc/os-release
  $ systemctl status libvirtd
  ```

- `centosbox`. `\centos` renders as **CentOS**

  On CentOS

  ```
  $ cat /etc/redhat-release
  $ systemctl status libvirtd
  ```

- `debianbox`. `\debian` renders as **Debian**

  On Debian

  ```
  $ cat /etc/debian_version
  $ systemctl status libvirtd
  ```

- `debianfamilybox`. `\debianfamily` renders as **Debian**, **Ubuntu**, or **Linux Mint**

  On Debian, Ubuntu, or Linux Mint

  ```
  $ cat /etc/debian_version
  $ systemctl status libvirtd
  ```

- `fedorabox`. `\fedora` renders as **Fedora**

On Fedora

```
$ cat /etc/redhat-release
$ systemctl status libvirtd
```

- gentoobox. \gentoo renders as **Gentoo**

On Gentoo

```
$ cat /etc/os-release
$ systemctl status libvirtd
```

- linuxmintbox. linuxmint renders as **Linux Mint**

On Linux Mint

```
$ cat /etc/os-release
$ systemctl status libvirtd
```

- redhatbox. redhat renders as **RedHat**

On RedHat

```
$ cat /etc/redhat-release
$ systemctl status libvirtd
```

- redhatfamilybox. redhatfamily renders as **RedHat**, **CentOS**, or **Fedora**

On RedHat, Centos, or Fedora

```
$ cat /etc/redhat-release
$ systemctl status libvirtd
```

- rpmfamilybox. \rpmfamily renders as **RedHat**, **CentOS**, **Fedora** or **openSUSE**

On RedHat, Centos, Fedora, or openSUSE

```
$ cat /etc/redhat-release
$ systemctl status libvirtd
```

- opensusebox. opensuse renders as **openSUSE**

On openSUSE

```
$ cat /etc/os-release
$ systemctl status libvirtd
```

- ubuntubox. ubuntu renders as **Ubuntu**

On Ubuntu

```
$ cat /etc/debian_version
$ systemctl status libvirtd
```

Modifying Distribution Titles

Sometimes you need to refine the title in the distribution box, perhaps to mandate a specific version. This is done as in the following example:

```
\begin{ubuntubox}[20.04]
   Something specific which needs to be done on Ubuntu 20.04
   \begin{cmdtt}
$ cat /etc/debian_version
$ systemctl status libvirtd
       \end{cmdtt}
\end{ubuntubox}
```

On Ubuntu 20.04

Something specific which needs to be done on Ubuntu 20.04

```
$ cat /etc/debian_version
$ systemctl status libvirtd
```

21.4 Custom Contextual Boxes

Contextual Boxes are useful when you want to make clear the environment (context) for the contained material, where things should be used or typed.

There is more than one class of macros, intended to support a variety of scenarios. You can use any of these context macros in any course; however, this is how they were intended to be used:

1. **Single computer** courses can use the `clilst` box to indicate things which are run on the command line.
2. **System Administration** courses often involve multiple computers or VMS can use the `onlaptop`, `onserver` or `onvm` boxes to indicate things which are run on different machines or environments.
3. **Embedded Development** courses involve a **host** machine and an embedded **target**. The `onhost`, `ontarget` or `serial` boxes specify things which are run in each of these contexts.
   The `serial` port macros might be preferred when showing things specifically on the serial console instead of over **ssh**. However, the `ontarget` box cab be used to indicate either a serial or **ssh** connection.

When more that one of these are used, it is worth using the default title boxes only on the first instance of the context box, which leaves only the icon to let people know the context of the box.

Multiple Context Box Usage

```
\begin{onlaptop}[The first laptop box]
  The frst laptop box
\end{onlaptop}
\begin{onserver}[The first server box]
  The first server box
\end{onserver}
\begin{onlaptop}[]
  The second laptop box
```

```latex
\end{onlaptop}
\begin{onserver}[]
  The second server box
\end{onserver}
\begin{onlaptop}[]
  The third laptop box
\end{onlaptop}
```

The first laptop box

The frst laptop box

The first server box

The first server box

The second laptop box

The second server box

The third laptop box

- Single computer

  `clilst`, and `\clifile`

  These are designed to show commands and output on the command line interface (CLI), and are particularly for courses which use only one machine.

  `clilst` will color-syntax highlight, but won't expand macros.

  On the command line

  ```
  $ grep root /etc/passwd
  $ sudo grep root /etc/shadow
  $ cd $HOME/LFT
  $ echo foo blah baz > foo.txt
  ```

  (If you wanted to expand `\course` you would have to wrap the code in `\begin{cmd}` and `\end{cmd}`.)

  `\clifile` is like clilst, but will allow you to read a command line session to show from a file.

  `file.cli`

  ```
  $ echo $PATH > doo.txt
  $ ls -l
  $ exit 0
  ```

- `onlaptoplst`, and `\onlaptopfile`

  Run these commands on your laptop (as opposed to on a server)

On Your Laptop

```
$ grep root /etc/passwd
$ sudo grep root /etc/shadow
$ echo \course
```

Very Important

onlaptoplst has problems with the use of #, the hash character at the beginning of the first line. To fix this you have to use a non-blank optional title as in:

```
    \begin{onlaptoplst}[On Your Laptop]
# grep root /etc/passwd
# sudo grep root /etc/shadow
    \end{onlaptoplst}
```

which renders as:

On Your Laptop

```
# grep root /etc/passwd
# sudo grep root /etc/shadow
```

There are other environments that have similar problems. If you do not want to use a title, you still have to insert a blank one such as [] to make things work properly.

onlaptopfile is just like clifile:

file.laptop

```
$ echo $PATH > doo.txt
$ ls -l
$ exit 0
```

- onserverlst, and \onserverfile
  These work similarly, and indicate the commands are run on your server (as opposed to on your local laptop).
  An example of \onserverlst:

On Your Server

```
$ systemctl status apache2
$ systemctl restart apache2
```

and an example of onserverfile :

file.server

```
$ echo $PATH > doo.txt
$ ls -l
$ exit 0
```

- `onvmlst`, and `\onvmfile`
  Run these commands on your virtual machine (VM).
  An example of `onvmlst` :

On Your Virtual Machine

```
$ sudo systemctl status apache2
$ sudo systemctl restart apache2
```

and an example of `onvmfile` :

file.vm

```
$ echo $PATH > doo.txt
$ ls -l
$ exit 0
```

Embedded Development Course Environments

The remaining macros in this section are only useful for embedded courses. Here, one must differentiate between commands to be executed on the **host** machine, or embedded **target**.

- `onhostlst` and `\onhostfile`
  An example of `onhostlst`:

On Linux Host Machine

```
$ ls /srv/bbb/root/
      $ sudo -i
```

and an example of `\onhostfile`:

file.host

```
$ echo $PATH > doo.txt
$ ls -l
$ exit 0
```

- `ontargetlst` and `\ontargetfile`
  An example of `ontargetlst`:

On Embedded Target

```
$ ls /srv/bbb/root/
$ sudo -i
```

and an example of `ontargetfile` :

file.target

```
$ echo $PATH > doo.txt
$ ls -l
$ exit 0
```

- `seriallst` and `\serialfile`

  Thses show output from the serial port. An example of `seriallst`:

Serial output

```
Boot up code

And yet more things.
```

and `serialfile`:

file.serial

```
This is example input.

This file has more than one line.
```

21.5 Listing boxes

When you want to show content appropriate to certain kinds of code listings (**bash**, **C**, **Python**, **YAML**, etc.), we have designed a number of specific **Listing Boxes** to give color sytax highighting to the contents.

Each type of listing has two actual macros, one for including content directly in the listing (with an optional title) and one for just including an external file in place. As the simplest example we have `filelst` and `fromfile{}`, which are used for any type of content that resides in a file.

Here are examples of these elementary macros:

- Show a generalized file snippet or the contents of a file with `filelst`:

```
\begin{filelst}[This is from a file!]
  This file has stuff in it.

  And yet more things.
\end{filelst}
```

which renders

This is from a file!

```
This file has stuff in it.

And yet more things.
```

- To just include a file in place:

```
\fromfile[My awesome title]{examples/file.txt}
```

rendering:

My awesome title

```
This is example input.

This file has more than one line.
```

For the remaining examples we will only show the result of the `*lst` macros as the results of the `*file{}` macros look the same.

Here are more examples, all of which follow the pattern we just showed:

- `clst` and `\cfile{}`

examples/file.c

```c
#include <stdio.h>

int main(int argc, char** argv)
{
   printf("Hello World!\n");
   return(0);
}
```

- `pythonlst` and `pythonfile{}`

file.py

```python
from mininet.net  import Mininet
from mininet.util import createLink

# create an empty network
net = Mininet()

# create all nodes in the network
c0 = net.addController()
s0 = net.addSwitch('s0')
h0 = net.addHost('h0')
h1 = net.addHost('h1')

# create the links between the nodes
net.addLink(h0,s0)
net.addLink(h1,s0)

# IP configuration for interfaces
h0.setIP('192.168.1.1',24)
h1.setIP('192.168.1.2',24)

```

```
# start the network
net.start()
net.pingAll()
net.stop()
```

• javalist and javafile{}

file.java

```
  public void init() {
        LOG.info("Initializing....");

        Preconditions.checkNotNull(dataBroker, "dataBroker must be set");
        Preconditions.checkNotNull(service, "dataBroker must be set");
        service.registerRpcImplementation(HelloService.class, this);
        WriteTransaction trans = dataBroker.newWriteOnlyTransaction();
        dataBroker.registerDataTreeChangeListener(DataTreeIdentifier.create(CONFIGURATION,
        ↪  GRT_REGISTRY), this);

        GreetingRegistry registry = new GreetingRegistryBuilder().build();
        trans.put(LogicalDatastoreType.OPERATIONAL, GRT_REGISTRY, registry);

        trans.commit();

        LOG.info("Completed initialization .....");
     }
\end{javalst}
```

• yamlist and yamlfile{}

kubeadm-config.yaml

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
kubernetesVersion: 1.16.1
controlPlaneEndpoint: "k8smaster:6443"
networking:
  podSubnet: 192.168.0.0/16
```

• shlst, and \shfile{}

file.sh

```
echo $PATH > doo.txt
ls -l
exit 0
```

• makelst and \frommakefile{}

Makefile

```
LATEX = pdflatex --shell-escape

%.pdf: %.tex
        $(LATEX) $< && $(LATEX) $< > /dev/null && $(LATEX) $< > /dev/null

$(DOCS): $(TEXFILES)
```

- `fdtlst` and `\fdtfile{}`
  This is for **Device Tree Files** used in embedded **Linux**.

fdtfile.dts

```
/ {
    node@0 {
        string-list-property = "string0", "string1", "string2";
        byte-data-property = [0x0A, 0x0B, 0x0C];
        reference-to-another-node = <&node1>;
    };

    node1: node@1 {
        more-properties = "abc", "def";
    };
};
```

- `latexlst` and `latexfile{}`
  Last and not least, code to show LaTeX source:

```
  \item \verb?fdtlst? and \verb?\fdtfile{}?

This is for \textbf{Device Tree Files} used in embedded
\textbf{Linux}.

\fdtfile[\texttt{fdtfile.dts}]{examples/fdt.dts}
```

Note that you should always have `\begin{latexlst}[]`. (Do not forget the `[]` or results will be unpredictable!)

22 Indexing

Indexing is Optional and Only Used in Some Courses

As of date **indexing** has been done for only some **LFD** development courses. So you can probably ignore this section.

Indexing is done as a relatively final step. As you are working on things if you can insert indexing commands that is great, especially as new material gets added. Students like indexes!

In LaTeX there is no useful automatic indexing program; as best I can see they don't work in any printing system anyway.

Simply embed the indexing commands as in:

LaTeX

```
\index{runtime alternatives}
\index{alternative()@\texttt{alternative()}}
\index{apply_alternatives()@\texttt{apply_alternatives}}
```

If you just want it in a normal type face you just embed in `\index{..}` tags. The other examples with the `@` show how to specify a typeface.

It is also possible to create sub-indexes which creates a new line under (for example) `kernel`:

LaTeX

```
\index{kernel!user mode}
```

Anyway, just look around the text of any class with indexing and you will see plenty of examples of all types.