Learning Materials — Style Guide

We are glad that you would like to help creating LPI Learning Materials!

This *Writing Style Guide* serves as an introduction to LPI Learning Materials' creation process and is supposed to be your initial source for technical advice. We aim to constantly improve this document to answer frequently asked questions. If you have questions beyond the information in this document, please contact us by emailing to learning@lpi.org.

Happy writing!

Your LPI Learning Materials Team

Introduction

LPI learning materials are created in a *single source publishing* process. This process turns plain text files into high quality publications. It is important that the documents you create follow the formal rules described in this guide.

To contribute to LPI Learning Materials process, you need to be familiar with three core technologies:

Markup

In your texts, you may not directly format individual elements. You can not directly work on the layout of your text. Instead, you should mark the structure of your text and let the publishing process turn this information into a proper layout. We use AsciiDoc(tor) for all our materials.

Editor

The only format used to work on LPI Learning Materials contents is plain text. The best choice for working on plain text files are text editors, such as Atom, vi, nano, Emacs. You should not use a complex word processor such as Word or LibreOffice. If you still do, make sure that you save your documents as plain text format without automatic text formatting.

Version control

We manage all texts and images of LPI Learning Materials in a version control system. We use Git on a server hosted by LPI. It is a great advantage to you familiarize yourself with the basic functions of Git and start using it from the very beginning. This ensures that all team members always have access to the most recent version. With *branches* we also use one of Git's core features. This means that each author has his/her own workspace (the assigned objective) within the project. Additions, changes etc. should only be made in this one assigned branch. Here you will learn more about this topic.

Length, Scope and Style

A lesson is approximately 3,000-4,000 words (20,000-30,000 characters) — including markup.

All technical terms, tools and technologies mentioned for the respective objective (under “key knowledge areas” and “partial list of the used files, terms and utilities”) must be mentioned and explained in the lesson.

The following “golden rules” apply to the presentation:

- Explain technical terms at their first occurrence.
- Develop explanations gradually and in the context of what has been said before.
- Find smooth, comprehensible transitions (and avoid bullet lists!).
- Give comprehensible examples and analogies chosen from the daily practice of the target group.
- Use the previously published Learning Materials on https://learning.lpi.org as a guide

Files and Structure

File Structure in the Git Repository

The structure of the Git repository and its branches is defined and should not be changed. After you have checked out the branch assigned to you (more information here), you will find the following file and directory structure (in the following example we use topic 5 and objective 5.2):

```
├──── 5
│    ├──── 5.2
│    │    ├──── 5.2_01
│    │    │    └──── index.en.adoc
│    │    └──── _index.en.adoc
 [...]
│    └──── _index.en.adoc
```

`5/`

The main directory for the topic that contains the subdirectories for the individual objectives.

`5/_index.en.adoc`

A text file with meta information about the topic that you should not change.

`5/5.2/_index.en.adoc`

The directory for “your” objective. Also here there is a metafile `_index.en.adoc` which you should not change.

`5/5.2/5.2_01`

The objective directory contains at least one subdirectory that contains all the contents of a lesson (here it is `5.2_01`, i.e. “first lesson of objective 5.2”). If the material of an objective is too extensive for a single lesson, it can be distributed over several lessons/subdirectories.

`5/5.2/5.2_01/index.en.adoc`

The lesson directory contains exactly *one* AsciiDoc file named `index.en.adoc`. In this file you write all contents according to the given text structure.

Images

If you integrate images into your text, create a directory `images/` in the lesson directory and save all images as PNG for the respective lesson, e.g.

```
|    ├──── 5.2
|    |    ├──── index.en.adoc
|    |    └──── images
|    |         ├──── image_01.png
```

Structure of a Text/Lesson File

We have already prepared a text file (`index.en.adoc`) in the first lesson directory of your objective. It contains the headings for the prescribed sections and looks as follows:

```
[[sec.5.2_01]]
= Lesson 1

[[sec.5.2_01-IN]]
== Introduction

[[sec.5.2_01-GE]]
== Guided Exercises

[[sec.5.2_01-EE]]
== Explorational Exercises

[[sec.5.2_01-SU]]
== Summary

[[sec.5.2_01-AGE]]
== Answers to Guided Exercises

[[sec.5.2_01-AEE]]
== Answers to Explorational Exercises
```

So you can start writing right away in any editor! The rules for the markup are explained in the following section.

Text Markup

LPI Learning Materials' publishing process is based on plain text files with a specific markup. Such files are used throughout the whole creation process. Their correct use is mandatory for everyone involved in the LPI Learning Materials creation.

The overall goal of the text markups is to enrich the text with information regarding its meaning — *not* regarding its layout. For example, a heading is a short text element that refers to the content of the following text or other sections and structures the text. This is unrelated to how this heading looks like: Position, color, size, spacing, etc. are irrelevant during writing and editing. These parameters are defined in the final steps of the publication process and depend on the individual target format.

All markups in the text are subject to some basic rules:

- Everything that structures the text according to meaning is directly marked up within the text.
- The markup is done with simple characters/strings. If you miss a markup element, please contact us.

This section describes how to use the allowed markup elements and macros. If your text requires other markup elements, please contact us. Do not use general markups, such as italics or bold fonts, instead.

Headings

Headings for chapters, sections, subsections, etc. are prefixed with equal signs. The number of equal signs marks the hierarchy level.

Hierarchy and structure of your working files is already given by the template: A lesson always starts with a heading of the 1st level, i.e.

```
= Lesson 1
```

This is followed hierarchically by mandatory and optional sections:

```
== Introduction
```

and your subsection:

```
=== A Smaller Part of the Introduction
```

Keep your text strictly structured. You may not skip any level: The “2nd level” is followed by another “2nd level” or at least two “3rd level”, but never by a “4th level”. On each level, you must have at least two headings; a section must have either no or at least two subsections.

Regular Text

Paragraphs are separated by one or several blank lines. The empty line(s) are converted to a paragraph break in the final layout. Between two words there can be any number of spaces. The additional spaces are converted to a single space in the final layout.

```
This is the first paragraph.

This is the second paragraph.
```

results in:

> This is the first paragraph.
>
> This is the second paragraph.

Inline Elements

Inline elements are defined within the regular text using “special” text elements. Please try to mark up your text as accurate as possible.

Emphases

Emphasized text is marked placed between underscores:

```
Marking _highlighted_ words
```

results in:

> Marking *highlighted* words

How exactly emphasized text appears in the final layout depends on the publishing process and cannot be defined within the text.

Please refrain from using these emphases where you can. Limit emphases to technical terms (but only at first occurrence) and necessary emphasis in the sentence.

IMPORTANT	Only use underscores as markup for emphases.

URLs

URLs which are complete, including a scheme (such as `https://` or `ftp://`), automatically become clickable without the need of any specific markup:

```
Please visit https://www.example.com for more information.
```

results in

Please visit https://www.example.com for more information.

To ensure a URL is not turned into a link, prepend it with a backslash (\):

```
The URL \https://www.example.com does not exist.
```

results in

The URL https://www.example.com does not exist.

To represent a URL by text, enclose the text in square brackets at the end of the URL.

```
please visit https://www.example.com[example URL]
```

results in

please visit example URL

Email Addresses

Email addresses work similar to URLs. An email address which is syntactically correct is turned into a link:

```
Please contact info@example.com for more information.
```

results in

Please contact info@example.com for more information.

Code and Machine Writing

In technical documentation, “machine writing” is important to mark up code snippets, command output, file and directory names etc. Any such text is placed in single backward quotation marks (“backticks”) within the regular text.

```
The file `myfile.txt` is stored in the directory `Mydocs`.
```

results in:

The file `myfile.txt` is stored in the directory `Mydocs`.

Quotation Marks

In typography, quotation marks are a science in their own — especially when it comes to different languages. That is why they are noted down in a way that avoids confusion and allows very specific conversions in the publishing process:

```
There are ``single'' and "`double`" quotation marks.
```

results in:

There are ‘single’ and “double” quotation marks.

Dashes

A dash is longer than a hyphen and it is represented by two hyphens (`--`).

```
A dash is longer than a hyphen -- and there's a reason for that
```

Keystrokes

The macro `kbd` is used to mark keys on the computer keyboard:

```
the key kbd:[Ctrl]
```

results in:

the key `Ctrl`

For key combinations, keys may be combined within the macro with a plus sign:

```
the key combination kbd:[Ctrl+C]
```

results in:

the key combination `Ctrl` + `C`

Block elements

Block elements are closed areas with their own formatting, separated from previous and subsequent text by blank lines.

Lists

There are three different list types: Item lists, numbered lists and definition lists. In general, lists are preferable to any form of table, as they can be transferred much more reliably to other formats.

Every item of a list starts with a capital letter.

Item Lists

Item lists, sometimes called “unsorted lists”, are useful to list elements without a specific sequential order. Each item is listed in an own line which begins with an asterisk *:

```
* Element A
* Element B
* Element C
```

results in:

- Element A
- Element B
- Element C

Numbered Lists

Numbered lists are useful for sequential steps, such as work instructions. Each step is listed in an own line, which begins with a period and a space:

```
. Step
. One more step
. And a last step
```

results in:

1. Step
2. One more step
3. And a last step

Definition Lists

Definition lists are useful for definitions of terms, parameters or other short explanations. Each term is followed by an explanation, separated by two colons:

```
Term A::
This is the detailed explanation of term A.

Term B::
This is the detailed explanation of term B.
```

results in:

Term A
: This is the detailed explanation of term A.

Term B
: This is the detailed explanation of term B.

Code Blocks / Listings

Code blocks are separated by dashed lines before and after the code:

```
-----
Here is the code
-----
```

It is best to wrap long lines of code “by hand” according to the rules of each programming language. A common maximum is 70 characters per line.

User input on the command line should be highlighted. To do this, write the line

```
[subs="specialchars,quotes"]
```

immediately before the block and mark the entries with an asterisk:

```
[subs="specialchars,quotes"]
-----
$ *ls*
file01.txt       file02.txt
-----
```

This results in:

```
$ ls
file01.txt       file02.txt
```

Provide only complete listings, including prompt and output, i.e. avoid (partial) listings or synopses which would not work as shown.

If the listing immediately follows an explanation as an example, do not use phrases such as “see below” or “see the following example” — the colon will suffice.

Here is an example:

```
The command `ls` is used to display the directory contents:

[subs="specialchars,quotes"]
-----
$ *ls*
file01.txt       file02.txt
-----
```

Illustrations

All image files should be located in the subdirectory `images` of the respective lesson. The images should be stored in the PNG format in good quality.

NOTE	Use images only for graphics or diagrams, not for screenshots! Text output e.g. on the command line should be marked as text or code blocks.

The general syntax for embedding an image into the text looks like this:

```
.<caption>
image:<path/filename>[id="<anchor>",width="<value>%"]
```

It includes the following placeholders:

- `<filename>` of the image including file extension (e.g. `.png`)
- `<caption>` in the shortest possible caption of the image
- `<anchor>` is used to refer to the image, as described below. The anchor must be unique in the text.

For example, the file `lpi-logo.png` could be embedded in the text like this:

```
The Linux Professional Institute logo.
image::./images/lpi-logo.png[id="fig.650",width="30%"]
```

results in:

Figure 1. The Logo of the Linux Professional Institute

The `width` of the image is specified as a percentage of the textwidth — it will be optimized later during the final text design.

Tables

NOTE	Please avoid tables where ever you can! Their complex and inflexible structure can not always be reliably converted into different layouts. In most cases, lists can include the same information. If you can not avoid using a table, the following minimum requirements apply:

Tables begin and end with the line

```
|==============
```

The number of equal signs is arbitrary, but it should be at least five.

Each row of the table is written in one line. The cells of a row are separated by the pipe character (`|`).

A caption is placed in front of the table in a separate line introduced by a period (without the following space!).

If you need a table header (i.e. a particular first table line), use `[options="header"]` after the table heading. The first line then becomes the table header.

Like a figure, you can also assign a unique anchor to refer to the table.

The following example:

```
[[tab.630466]]
.caption
[options="header"]
|============================
| Column 1 | Column 2 | Column 3 | Column 4
| 1.a | 1.b | 1.c | 1.d
| 2.a | 2.b | 2.c | 2.d
| 3.a | 3.b | 3.c | 3.d
|============================
```

results in:

Table 1. caption

Column 1	Column 2	Column 3	Column 4
1.a	1.b	1.c	1.d
2.a	2.b	2.c	2.d
3.a	3.b	3.c	3.d

Note and Warning

Note and warning are rendered in boxes. Don't overuse them and make sure that they either highlight either dangerous circumstances or relate to common pitfalls.

Each box belongs to a specific class which indicates the type of the box. The following types are available:

`[NOTE]`
: for a (mostly practical) note in connection with more theoretical explanations

`[WARNING]`
: for warnings that are important when using or omitting described actions

The type is specified in square brackets. It is followed by an (optional) heading, introduced by new line starting with a dot. The box content follows between two lines of equal signs. The syntax for these blocks is as follows, including an optional heading:

```
[NOTE]
.Heading
==========================================================
This is particularly important.
==========================================================
```

results in:

NOTE *Heading*

This is particularly important.

Quotations

Longer quotations are separated by lines of underscores; the type and details of the author and source are shown in square brackets, separated by commas:

```
[quote, Eric S. Raymond, The Cathedral and the Bazaar]
--------------------------
The idea of open source has been pursued, realized, and cherished over those
thirty years by a vigorous tribe of partisans native to the Internet. These
are the people who proudly call themselves "`hackers`" -- not as the term is
now abused by journalists to mean a computer criminal, but in its true and
original sense of an enthusiast, an artist, a tinkerer, a problem solver,
an expert.
--------------------------
```

results in:

> The idea of open source has been pursued, realized, and cherished over those thirty years by a vigorous tribe of partisans native to the Internet. These are the people who proudly call themselves “hackers” — not as the term is now abused by journalists to mean a computer criminal, but in its true and original sense of an enthusiast, an artist, a tinkerer, a problem solver, an expert.
>
> — Eric S. Raymond, The Cathedral and the Bazaar

Comments

A comment is text that is not rendered later, i.e. is not to appear in later output formats. Longer quotations are separated by lines of slashes (//////).

Please use comments very sparingly (preferably not at all!).

Exercises and Answers

Each lesson includes the sections “Guided Exercises” and “Explorational Exercises” and the corresponding answers. To ensure that these sections can be used by both teachers and learners in the same way, we ask that you observe the following markup guidelines:

Numbering Exercises

The exercises in a section are numbered consecutively. Place a dot (.) in front of the question, as described in section Numbered Lists.

```
. This is the first exercise
```

Space for Possible Answers

Leave space for possible answers directly below an exercise so that the exercise part can be used

appropriately in class. Depending on the response options, the following markup variants are supported.

Single-line response field

Use a single-line, empty table for this purpose

```
. This is an exercise.
+
|=====
|
|=====
```

Place a line with a plus sign (+) between the exercise and the response field so that the block is rendered as a whole.

results in:

1. This is an exercise.

Multi-line response field

Use a multi-line (maximum 4) table if one comprehensive or several short answers are expected.

```
. This is an exercise.
+
|=====
|
|
|
|=====
```

results in:

1. This is an exercise.

Multi-column response field

Use a multi-column table if unique assignments are to be made

```
. This is an exercise.
+
|=====
| Item 1 |
| Item 2 |
| Item 3 |
|=====
```

results in:

1. This is an exercise.

Item 1	
Item 2	
Item 3	

Answers Sections

- Make sure that you repeat the exercise in the exact wording.
- Write the sample solutions directly below the exercise and connect question and answer(s) with a plus sign (+).
- Write answers with continuous text as normal paragraphs.

```
. This is an exercise.
+
This is the sample solution to the exercise as continuous text.
```

results in:

1. This is an exercise.

   This is the sample solution to the exercise as continuous text.

- Markup answers in the form of commands or other shell inputs and outputs like other listings.

```
. This is an exercise.
+
[subs="specialchars,quotes"]
-----
$ *command*
-----
```

results in:

1. This is an exercise.

```
$ command
```

- For unique assignments, use a multi-column table:

```
. This is an exercise.
+
|=====
| Item 1 | Answer 1
| Item 2 | Answer 2
| Item 3 | Answer 3
|=====
```

results in:

1. This is an exercise.

Item 1	Answer 1
Item 2	Answer 2
Item 3	Answer 3

Version Control

General information

All project files are stored and managed centrally in a so-called *Version Control System* (VCS). Using a VCS has numerous advantages, including:

- All project members have access to all files at all times.
- The files are always up to date.
- The history of all changes is tracked and allows reverting to former versions.
- The files are always in a consistent state and are regularly backed up.

The VCS we use is Git. Git is widely used and the base functionality is easy to learn.

You will receive access to the Git repository of your projects from the LPI Learning Material Team. LPI hosts its Git repository using a self-hosted platform called GitLab.

Working with Git

First Login

When you first work with LPI's GitLab, you will receive an email with a link to a website where you can enter your password.

Then you can either connect to the server via the GitLab web interface or via the command line using the command `git`. Usually, you will work on the command line to manage the files you edit on your computer. This section introduces you to the most important commands on the command line. We assume that you have installed git on your computer for this purpose; the easiest way to check this is via

```
$ git --version
git version 2.10.1
```

If the command does not return a version, you have to install git first.

Configuration

To avoid having to enter your user name for every `git` command interacting with the server, it is recommended that you store this in your local git configuration:

```
git config --global user.name "YOUR_USERNAME"
```

Download Working Copy

The repository you're using will already be set up when you get access to it. At the beginning, you have to create a local working copy of the repository. Use the command `git clone` to download a copy of the repository:

```
$ git clone https://git.lpi.org/<name_of_the_project>/<name_of_the_repository>.git
```

The resulting directory is now your working directory.

Working with Branches

As already mentioned at the beginning, each author has his/her own workspace within the project for his/her objective. In Git such a strictly separated workspace is called *branch*. The following steps are advisable and essential to work just within your branch:

1. To check which branch you are currently in, use the command:

```
$ git branch
...
  3.3
  4.1
  4.2
  5.2
  5.3
...
* master
```

So you are in the branch `master`, but you should not make any changes in it!

2. So change to your personal branch, e.g. `5.2`:

```
$ git checkout 5.2
git checkout 5.2
Switched to branch '5.2'
Your branch is up-to-date with 'origin/5.2'.
```

You are now in your branch and can start working on the text.

Add and Edit Files

Maybe you will add new files to your project, be they text (plain text files with the extension `.adoc`) or images (in `.png` format in the `images` directory). Each of these files must first be "registered" to Git:

```
$ git add newFile.adoc
```

You have informed Git that you have added the file `newFile.adoc` to the project, but in order for this file to be really version controlled, it must be "officially committed" to Git in another step. Even files that are already under version control and have only been modified by you must first be "registered" (`add`) and then "passed over" (`commit`).

```
$ git commit -m "5.3_01: new file about XY" newFile.adoc
```

The `-m` option allows you to add a comment to your change, which is very useful for keeping track of the progress of the project.

For a better overview, please precede the comment with the number of the project or subproject (i.e. the Objective/Lesson).

This information is now stored internally by Git, but only in your local work system. In a final step they have to finally transfer the changes you have planned to the server so that all other project participants can see them as well. Of course you can register many work steps with Git and transfer them to Git during a work session and then transfer them to the server:

```
$ git push
```

Update working copy

Before you start a working session, you should update your local working directory, i.e. transfer all changes made by other project participants to your working environment:

```
$ git pull
```

Writing Conventions

Headings

- Capitalize “important” words in a heading (nouns, pronouns, verbs, adverbs, and more)
- Do not capitalize “short” and “less important” words (articles, prepositions and more)

(Cross) References

The lessons are designed as self-contained units, which can — also in printed form — ideally be used without further sources or network access. Therefore:

- Cross-references within a lesson only through general formulations such as “see above” or “as explained above”, *not* through links.
- References to other lessons also only through general hints (e.g. “Details can be found in the lesson on Using the Command Line”)
- References to external/further sources are best in the form of a `TIP`. These should then be the website of a project or a page authorized by the project, i.e. *no* references to general sources such as Wikipedia, Stackoverflow or similar.

Fictional Usernames

Don’t use your personal (user)name for examples or in shell output. Please use the following fictional usernames for your examples:

```
Carol
Dave
Emma
Frank
Grace
Henry
```

Special Terms and Expressions

The following list mainly contains technical terms that should be used and written as uniformly as possible throughout the LPI Learning Materials.

Please send us further questions or cases of doubt so that we can continually add to the list.

B

block device

C

character device

E

email

F

filename
filesize
filesystem
filetype

G

gigabyte

K

kilobyte

L

login (noun)
log in (verb)

M

megabyte

```
S

soft link
subdirectory
subprocess
```

```
U

Unix
```