Packt Chapter (H1 – Chapter)

Enter your chapter introduction here. It shouldn’t exceed much more than 300 words. Introductions should do the following (P – Regular): Introduce the topic or topic areas we’re going to cover; Tell the reader exactly what they’ll practically be doing, achieving, and learning in the upcoming lessons or activities; tell the reader why these lessons are useful. What will readers be able to do by the end of the chapter? This is your opportunity to outline a value proposition to the customer.

Add the line, “In this chapter we’re going to cover the following main topics:” Then, add a bullet list of your main chapter headers. Your main headers should denote the main topics or tasks covered in the chapter. The purpose of this bullet list is to allow readers to easily navigate to certain sections.

* Main topic 1 (L – Bullets)
* Main topic 2
* Main topic 3

Technical requirements (H1 – Section)

In (P – Regular) this section, add the technical requirements for your chapter. List the technologies and installations required. You’ll also need to provide the Github URL, for example, <https://github.com/PacktPublishing/Getting-Started-with-TensorFlow> (P – URL), for the code in the chapter. Create a Github folder named, "chX", where X is the chapter number. For example, ch1.

Header 1 - Main topic or main task (H1 – Section)

The (P – Regular) language you use in your header titles should always indicate what the reader is going to do or learn in the following section. This will nearly always mean including a verb, preferably in the present participle or gerund ‘-ing’ form, for example: creating x, building y, implementing z (creating an environment, building a stack, implementing software design principles).

The opening paragraphs or the opening few sentences of each section should discuss what the reader is about to do/achieve. Your opening should also mention, if it isn’t implicit, why the lessons are applicable and useful. If the section involves completing a task, you might need to discuss the method.

Header 2 - Subtopic or subtask (H2 – Heading)

Let’s (P – Regular) look at how to present code in your chapter. Make sure your source code is working, then paste it into your Word template and format it in Packt code style. Don’t type your code directly into the document editor, as capitalization and auto correct can break it.

We need to ensure that every code block and image is directly preceded by a “lead-in.” A lead-in sentence tells readers what the following code is or does. It’s important that they know this information before, not after seeing the code/image. You should also tell readers where the code should be placed, if it’s not already clear. For example, the file might need to be in a certain folder or be in the event handler for a particular button before it will work. Use SC – Source to add the code block format. If you need to highlight any piece of the code block, use SC – Highlight like it has been done for function (P – Code for code in text) in the code block below.

const set = function(...items) {  
 this.arr = [...items];  
 this.add = function(item) {  
 if( this.\_arr.includes(item) ) {  
 return false; (SC - Source)

Further explanation or discussion of how the code works can come immediately after your code block.

Given the practical nature of Packt lessons, you’ll often want your readers to run code, so your lead-ins will be written as instructions. These instructions should denote what the code will achieve, rather than simply state “run the following code.”

Important note

Avoid using comments in code. The best way to format code explanation or instruction is as regular body text. Comments tend to get lost in code blocks. They're not immediately distinguishable from the code, they can end up being poorly formatted, they may contain spelling mistakes that are difficult to correct, or they may be written in a shortened form that is difficult to interpret.

The closing paragraph or sentences of sections shouldn’t end abruptly end with code/a screenshot or simply ‘this completes the task.’ Readers require a summarizing sentence or two - a signpost. This signpost should reiterate what skills they now have, or mention how what they've just done/learned links to the next task or the overarching purpose of the chapter.

Header 2 - Subtopic or subtask (H2 – Heading)

Let’s (P – Regular) look at how to present commands in your chapter. Like with code, make sure the commands are working, then paste it into your Word template and format it in Packt command style. To format command lines, use P – Source. Make sure to add a lead-in telling readers exactly what we are doing with the command.

$ apt-get install node.js

$ apt-get uninstall node.js

Make sure to explain what the command will do. If readers are meant to take action based on the output, show the output as part of the command block above or use a screenshot and then discuss what they should look for.

Header 1 – Main topic or main task (H1 – Section)

Use (P – Regular) numbered steps for sequential instructions. Remember, readers will often be practically implementing your lessons, looking between your content and their own computer screen. Numbered steps help them to keep their place.

1. Step one. (L – Numbers)
2. Step two.
3. Step three.

If you need to add paragraphs under specific numbered steps or if you need to use a block of code or commands, use L – Regular for paragraph text and L – Source for the code or commands. You can use these options to add paragraph text and code or commands under specific bullet points as well.

1. Step one. (L – Numbers)

This paragraph appears underst step 1.

1. Step two.

const set = function(...items) {  
 this.arr = [...items];  
 this.add = function(item) {  
 if( this.\_arr.includes(item) ) {  
 return false; (SC - Source)

1. Step three.

Summary

In your Summary section, reiterate the key lessons covered and skills learned in the chapter. Remind readers why these lessons or skills are useful. Finally, add a sentence or two on what we’ll be covering in the next chapter, highlighting how this is the next natural step from what we’ve just covered.

Here’s a set of visual aids for other styles as well.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#2 This is how P – Bold style will look:

This chapter reviews the many ways Wireshark can filter traffic.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#3 This is how P – Italics style will look:

To help your learning of the different ways to refine your view.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#4 This is how P – Keyword style will look:

We'll cover when to filter Traffic and outline the difference between display and capture filters.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#5 This is how P – Code style will look:

So that you can refine your java.exe file when filtering traffic.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#6 This is how P – URL style will look:

We'll review ways to create more complex filters (https://www.google.com) by using the expression builder.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#7 This is how P – Source style will look:

apt-get install node.js

apt-get uninstall node.js

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#8 This is how P – Callout style will look:

We'll then go through capture filters and how they use syntax that is different than display filters. Finally, because filters are so handy, we'll cover some tricks, shortcuts, and common filters that will help you achieve a more effective analysis.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#9 This is how P – Callout Heading style will look:

Important Note

While in the course of your daily routine, the network starts to experience a significant slowdown. You check your Intrusion Detection System (IDS) and anti-malware protection, and there is no evidence of intrusion. At that point, you grab a quick capture to determine the source of the slowdown. Wireshark, along with many other packet analysis tools, has the ability to take a large capture, filter on specific traffic, and refine your view to help with analysis.

Always use P – Callout Heading and P – Callout together.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#10 This is how L – Bullets style will look:

* Filtering network traffic
* Comprehending display filters

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#11 This is how L – Numbers style will look:

1. Creating capture filters
2. Understanding the expression builder

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#12 This is how L – Regular style will look:

* Discovering shortcuts and handy filters

Filtering network traffic

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#13 This is how L – Source style will look:

* Wireshark has several options to filter traffic:

Display filters: Used during an active capture or on a pre-captured packet

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#14 This is how SC – Source style will look:

Capture filters: Applied prior to capture to only display a certain type of traffic

Expressions: Creates complex filters using logical operators

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#15 This is how SC – Highlight style will look:

When filtering traffic, there is a difference between display filters and capture filters. In the next section, let's explore the difference.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#16 This is how SC – Heading style (Filename above code snippet) will look:

Cjava.py

When working with packet captures, it appears as if the capture and display filters are the same. However, although the two work in similar ways, capture and display filters each use their own syntax.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#17 This is how SC – Link style (GitHub Link below the code snippet) will look:

While using Wireshark, there are four main phases of packet analysis as discussed in [Chapter 2](https://cdp.packtpub.com/learn_wireshark___fundamentals_of_wireshark_/wp-admin/post.php?post=30&action=edit#post_25), Using Wireshark NG, which are Gather, Decode, Analyze, and Display, as shown in the following diagram:

https://github.com

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#18 This is how IMG – Caption style (Figure and Figure caption) will look:



Phases of packet analysis

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#19 This is how H1 – Section style (Heading 1) will look:

Gathering network traffic

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#20 This is how H2 – Heading style (Heading 2) will look:

The packets pass through the appropriate capture engine

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#21 This is how H3 – Subheading style (Heading 3) will look:

NPcap or WinPcap

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#22 This is how H4 – Subheading style (Heading 4) will look:

Capture filters

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#23 This is how P - Quote style will look:

“use the Berkley packet filter syntax, and when used, Wireshark drops any packets that are not in the filter. You can read more about this in The BSD Packet Filter: A New Architecture for User-level Packet Capture”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#24 This is how SP – Editorial style (Heading 1) will look:

Please do not delete this paragraph.