CHAPTER 5

Philology: Standards for Digital Editing

Among the most attractive aspects of digitized manuscripts is their use for philological purposes. When setting out to edit a text, you will invariably find that its manuscript copies are scattered over different libraries, and it is costly and impractical to visit them one after the other. Indeed, for most of the twentieth century, scholars already collected photographs or microfilms of their manuscript copies in order to lay them side by side and work on them without the pressure of opening hours or erratic viewing policies at libraries. In the digital age, bringing surrogates together in one digital environment is a godsend. All too often, discussions about digital critical editing are about its end result and the thorny question whether or not we should be making a 'digital edition.' Since that topic is worthy of a handbook in itself, we will not go into it in great detail. Rather, we shall look at our digital workflow in a more general way, paying attention to the standards we ought to use and how the photos of digitized manuscripts can fit in that workflow. We start with a general introduction to standards in the world of computers. Then we discuss how to move from simply typing text to building a critically marked-up text, and finish that section by discussing several typical use cases. Then we look at how images can be incorporated into the workflow. We finish with notes on how to preserve and continue to work on the materials we produce along the way, and finally revisit that delicate issue of digital editions.

A standard is nothing more than a detailed agreement to do something in a particular way. Abiding by this agreement is important because if large amounts of people do things in the same way, they can benefit from each other. Living on the other side of the Atlantic will make it rapidly clear just how important standards are—for length, weight, temperature, shoe size, where to place a comma in a number, how to display afternoon time … the list goes on.

People can easily spot if something is off, but a computer cannot. If I say the temperature is 20 degrees and it is summer, an American will understand I am talking in Celsius, not Fahrenheit. A computer controlling a heating system, however, will take that input without complaint, turning the system at full blast to combat the supposed freezing conditions, if it is set to Fahrenheit. If a person controls the input to the system, there is a good chance the person will convert the temperature to Fahrenheit before putting it in, but if it is only another computer that hands over the temperature input (say, the server of a

© CORNELIS VAN LIT, 2020 | DOI:10.1163/9789004400351_007

national meteorology institute), there will be no possibility at both ends of the communication to intelligently evaluate the message. And here is the catch: most of the times in our work, we talk to each other by having two computers talk to each other. If I use a messaging app, on a technical level, I am not directly communicating with someone else, but I am relaying keyboard inputs to my smartphone, which in turn talks to someone else's smartphone, which then relays differently colored pixels to that other person. I once tweeted the emoji for the Kaaba, the black box-shaped building in Mecca toward which Muslims pray. From people's responses, it became clear that they did not have this emoji installed, and their computer could, therefore, not display it as a little depiction of the Kaaba: 🕋, but instead displayed the standard sign for a character that is not available, which is a black square with question mark like this: �. Obviously, that is not a picture of the Kaaba, but in a moment of digital poetry people thought that this square with a question mark is an excellent symbol of it. You will probably have concluded by now that it is of the greatest importance that computers run their operations by strictly defined yet broadly accepted standards.

We will look at old standards and new standards, open standards and closed standards, standards defined by the tech industry, and standards defined by academic stakeholders. There will be standards that you consciously use daily, those that do their work under the hood of your computer, those that you may not need to bother with ever. We will even get a feeling for good and bad standards. In this chapter, we will go through virtually all parts of a typical workflow, proceeding from the smallest, most computer-related scale of bit and byte to the widest, most human-related scale of publish and cite.

1 File Formats

We can think of words, sounds, and images in the abstract, but we also know that when we want to embody them, they require a certain form. That form, in turn, dictates the way it is supposed to be used. If music is recorded on vinyl, you cannot push it into a CD player and expect to hear the music, and a manuscript is used differently from a scroll, in order to read the entire text. It is the same with files on a computer: they are in a certain format, which stipulates rules for a program as to how to open the file correctly. For example, if you tried to open a .jpg file with a simple text editor (like *Notepad* for Windows or *TextEdit* for macOS), you would not see the image visually, but you would see a seemingly random string of letters and other characters along multiple lines.

That is because the text editor expects the file it is opening to be a text, and it will convert whatever bits and bytes the file is made of into the text displayed on the screen.

In the world of computers, an important distinction is made between text file formats and binary file formats. For text, you may be thinking of a few notable file formats such as .txt, .doc (and .docx) and .pdf; however, of these examples, only .txt is a text file format, and the others are binary file formats. All text file formats admit to two restrictions: (1) they only contain plain text (written characters broadly understood) and (2) this text is organized in lines separated by a *new line* character. In a happy moment of agreement, all operating systems can open text files natively and opening any sort of text file in any sort of text file application will display the contents of the file. Text file formats are mostly defined to hold textual or numerical data (examples are .txt, .xml, .json, and .csv) or to contain code (examples are .py and .js). This is in stark contrast with binary files, of which each has a unique way of storing their data. There are binary file formats that are quite common, such as .jpg or .mp3, but many more file formats can only be written and read by a particular program. In fact, many of them are proprietary, and the particular way they encode data is a trade secret jealously guarded by the company which owns it. For example, .mellel is a closed, proprietary format used by the macOS word processor Mellel, meaning that only the owners of the file format know exactly how it works. Opening it in a plain text editor will result in a garbled mess.

The most important text file formats are .txt and .xml, on which more will be explained in the section about mark up. Microsoft Word's .doc is a binary file format. This is due to the particular way Microsoft has encoded the layout and markup possibilities for a Word document. For years, its encoding was not disclosed. Its successor, .docx, is a strange mix of text and binary: on its own, it is a binary file, but if you change the .docx extension to .zip and unzip it, a collection of files and folders become available that are mostly text files in the form of xml. The same cannot be said of Adobe's .pdf file format, a big favorite in the humanities for its function as a print-like digital file,[1] which is squarely a binary file format. When opened in a PDF viewer, everything looks nice and shiny, but under the hood, .pdf can be quite a mess. Therefore, even from PDFs with digital text (i.e., selectable and searchable), it is not always easy to extract the text automatically. Here, we stumble upon a practical difference between text files and binary files: text files are by far preferable when we want to do

1 For as long as (and long after) the humanities are figuring out the transition to the digital world, especially as long as there is fear that publishing digital-only 'does not count', I suspect that PDF files will function as a hybrid: they are existentially born-digital but with the look and feel of a print material.

complicated or automated manipulations on the file, but even though binary files carry more bloat with them, they can present the user with a rich experience right out of the box. The .pdf format is a good example of this. By saving something in .pdf, you can be assured that when you share the file, the other person will see it in exactly the same way as you created it, and you will also be assured that the file cannot be (easily) altered. Compare that with sharing a .doc (or .docx) file. Perhaps you created the document with a particular font that the other person does not have on their computer. In that case, when that person opens the document, Microsoft Word will use a different font to render the text, and this may cause all the lines and paragraphs to shift slightly. If there is a complicated layout with tables, footnotes, and images, this may not line up anymore.

Text file formats are not entirely without markup. For example, spaces, punctuation, and indications of where a line ends (and a new one begins) are native to text file formats even though we might not strictly classify them as plain text. This is especially true for the *new line* marker, which can be inserted easily by hitting the *Enter* key on your keyboard. We often do not think of it as a character, yet it is one of the most frequently exploited characters in programming, as we can give a command to do something for every string of text we find on a separate line. *New line* also allows .csv files (comma separated value) to store a primitive table in which each line represents a row, and every column is separated by a comma. Codes, such as .py for Python or .js for JavaScript, are also text file formats, and some programming languages see the *new line* as an indicator for a new command to be executed.

Text file formats form a good mix of being human-readable and machine-readable. This is why they are excellent for storing code and textual and numeric data. Programmers use text file formats all the time. Meanwhile, binary files are better suited for presenting rich media and, therefore, they are the kind of file format that is more often encountered by the general public. We in the humanities will have to straddle both worlds. For example, we can get far by using digital photos of manuscripts in a conventional manner by simply loading them in an application that displays the image on our screen. If we want to go further, as we will do in Chapter Seven, we basically have to turn the image (a binary file) into a text format, by reducing the image to a few million triple values representing the mix of red, green, and blue for each pixel. Only then will we able to do computations and automated transformations on it.

Images are, of course, of special interest to us. They can come in a great variety of formats. The most fundamental digital data of photos is called a raw image file, and this can take a great variety of file formats, depending on the factory and the type of camera you are using. This cannot be used in and of itself for viewing and manipulating. Instead, from here, two file formats are often used: .tiff and .jpg. The former can store the image in lossless quality, meaning that no compression is applied that would reduce the quality of the raw image. The latter has variable compression, allowing for a trade-off between file size and quality, which is especially attractive if the images need to be available on the internet. In Chapter Four, I already introduced vector images as a different kind of image. Rather than storing an image pixel by pixel, such images store objects mathematically by begin point, end point, and the curve of the line in between (and from there, other aspects such as line style and fill). Since these images are defined rather than prescribed, their actual rendering on a screen is done on the fly, which ensures that the image is always sharp and crisp, no matter how far you zoom in. Obviously, this format is not suitable for photographs, but more so for digital drawings. An interesting aspect of vector images is that they can be stored in text file formats. One such format is .svg, which is often used on websites. Perhaps surprising, but .pdf is principally also a vector image file format, especially suited for print or print-like purposes. As we have seen, pdf is a binary format.

It is an essential skill to identify the right file format for the job at hand and, by extension, it will be very advantageous to be able to convert files from one format to another. We can distinguish four levels to do that: one is to use the functionality provided by the operating system itself, another is to use an application with a graphical user interface, the next is to use an application that can only be used with a command line in the terminal, and the final option is to program a script yourself.

For simple image manipulation, a myriad of applications can help you. I have found Resize Sense (macOS only) to be a useful application with a GUI to do batch rotating, resizing, and renaming, for example, after I have taken a few hundred photos of a manuscript. ScanTailor is a wonderful tool for page splitting, auto rotate and dewarping, and (if desired) turning photos into sharp black and white images.[2] Adobe Acrobat Pro can be used to extract all pages as individual photos from a PDF or create a PDF from individual photos. Occasionally, for example, if you receive photos in an unusual file format or if they have an unusual size, you might have to resort to the very powerful ImageMagick, which you can instruct to do things from the command line.

The same goes for taking data from the internet. You might just simply click a button that says 'download' or you might right-click and select *Save As*. You might also have to go dig deeper using your browser to find the page resources.

2 This is especially useful for sources that are originally in black and white, such as printed publications. Returning them to black and white will often times make them more legible and smaller in file size. If there is good OCR software for the script the text is written in, this would be an excellent preparatory step.

If that is too tedious, perhaps because you are after a whole number of files, you could use a plugin for your browser with a GUI, or a stand-alone application called a download manager. Through all kinds of technical features, these applications can truly reduce the time to download something. They can also stop and resume downloads and make sure you are not banned for overusing, among other things. A very powerful but command line only software is *wget*. And lastly, there is the possibility to fine-tune it exactly as you want or need with web scraping tools and coding it yourself.

It is best practice to slowly escalate the kind of tool you use. Usually, this is dictated by the limitations you run into, and in this sense, you will notice soon enough when you need a different approach like a command line tool or a self-coded solution. Let us consider one example here. Say I shot eight photos with my iPhone, and I now want to make one PDF of them to share it with someone else. Those eight files are shot in .heic file format and amount to 9.1MB on disk. Once I have them on my laptop, I can convert them in several different ways, which I briefly summarize below, with the method on the left and the file size of the resultant PDF on the right:[3]

TABLE 5.1 Different results for converting an image

Native support of the operating system	
Right Mouse Click > Quick Actions > Create PDF	131,7MB
Opened in Preview in one window > Print > As PDF	131,7MB
Using Automator 'New PDF from Images'	130,1MB

Using an application with GUI	
Using iMazing HEIC Converter for conversion to .jpg, then Right Mouse Click > Quick Actions > Create PDF	39,2MB
Using Adobe Lightroom for conversion to .jpg, then Adobe Acrobat for conversion to .pdf	25,1MB
Using FreeToolOnline-website	919KB

3 The data produced here is, of course, heavily dependent on the exact version of hardware and software. I did this test with the following equipment: iPhone 7 Model A1778, MacBook Pro Retina 13" Mid 2014. Software: macOS Mojave 10.14.1, iMazing 1.0.7, Adobe Photoshop Lightroom CC 2.0.2, Adobe Acrobat Pro DC 2019.008.20071, http://freetoolonline.com/heic-to-pdf.html accessed on 11-11-2018, ImageMagick 7.0.8-14.

TABLE 5.1 Different results for converting an image (*cont.*)

Using a command line tool	
Using ImageMagick for conversion to .pdf	295,1MB
Using ImageMagick conversion to .jpg, then to PDF	16,3MB

On macOS, the most direct way, especially if you are only concerned with a quick result, is to right-click on the photos and select from the dropdown menu the submenu *Quick Actions* and then click on the option *Create PDF*. This will create a PDF file in the same folder as the photos are and will put the focus on the file name so that you can immediately type out a different file name. Though it is a convenient way, you end up with a file that is extraordinarily large. You could also open all the photos in Preview, the standard image viewer of macOS, drag their thumbnails all into one window, and then simply print as PDF, but this results in exactly the same file size, this time with some white space on some edges. A slightly more advanced approach but still using the functionality of the operating system only is to use Automator and its function *New PDF from Images*. This gives, a bit surprisingly, a slightly (but not much) smaller file size.

Clearly, only in the direst of situations and only if you need to create one PDF would this be an acceptable solution,[4] so the next step is to use a more specialized program with a point-and-click graphical user interface. The free application iMazing HEIC Converter helped me out here. It allows a quick drag-and-drop of .heic files and converts them to .jpg. Of course, that does not make us a PDF, so we still need to use the function native to macOS to *Create PDF*. iMazing manages to convert the HEIC files to JPG, but it does so at the cost of increasing the file size fourfold. The upside of this workflow is that once they are in JPG format, the *Create PDF* function adds no extra file size to them. iMazing is on the easy and cheap end of the spectrum of consumer software, so let us see how software on the other end of the spectrum fares. The software suite of Adobe is the obvious candidate, whose sticker price is beyond most mortals' reach, but most universities will likely offer it for free to students and employees. The application for creating PDFs, Acrobat, does not support HEIC files, so we will first need to use Lightroom to convert the files to JPG.

4 One way of sharing such a file is to place it in your Dropbox (or equivalent) account and share a link.

Combining both applications will result in a PDF that is significantly smaller than the previous workflows, all the while retaining quality.[5]

An entirely different approach is to use an online resource to do the conversion for you. Though a few websites advertised support for HEIC files, they actually did not. In the end, 'FreeToolOnline' did work: I only had to visit the website, upload the images, and click 'convert.' This gave a shockingly small file size, small enough in fact to attach to an email, but needless to say, this was at the cost of reduced image quality. There are no options to control this, and the reliance on a website is itself cumbersome.

A final resort is the command line tool ImageMagick. However, its immediate conversion to PDF did not fare well: not only did it result in an absurd file size, but the colors were completely off. Converting to PDF through a middle conversion to JPG gave a very satisfactory result, with the smallest file size and no compromise in resolution.[6] It should be noted that in fact, all the methods described here resulted in slightly different colorations, which will be a high priority issue for curators but is less of a concern for our private usage. For example, when doing a visual comparison, ImageMagick's JPGs were slightly lighter than the original HEICs, while Lightroom's JPGs were slightly darker than the originals. We see, then, that the different processes we can apply to convert file formats will lead to different results. Being aware of the possibilities, then, is very useful.

Let us do a similar comparison for text. In this example, I started with a Word document, containing an introduction and edition of a short 17th-century epistle. The document contains English, Arabic, and Persian, with a combination of footnotes and endnotes.

TABLE 5.2 Different results for converting a text

Method	**File size**	**Correct characters**	**Foot-/Endnotes**
Original .docx file	177kb	yes	yes
Using Microsoft Word, saving as .txt	79kb	slight mixup	only text
Using Pandoc	81kb	yes	yes
Using Python library 'python-docx'	66kb	slight mixup	no

5 Lightroom changed the PPI from 72 to 240. This obviously does not come with quality increase, but does increase the file size and gives a false impression of higher quality if one were to 'fly by instruments.'

6 I first used this command: magick convert *.HEIC -set filename:f '%t' out/'%[filename:f].jpg' Then this: magick convert *.jpg manuscript.pdf

If we simply open the file in Word and click File > Save As …, we can save the file within Word to a .txt format. The line breaks are preserved, and so are the indentations at the beginning of every heading and paragraph, but no further distinction for headers or any other formatting is made. There is a slight mix-up when the Arabic and Latin characters are mixed, but more importantly, while the text of the notes has been preserved, there is no reference to where they belong in the text.

Pandoc is a powerful command line tool to transfer different text formats.[7] It also supports TEI-XML, a standard we will soon discuss. With this .docx file, it creates a .txt file that is remarkably useful. Virtually all formatting is preserved: for example, headings are distinguished by a following line with either = or - signs, italic text is indicated with a * sign, and the notes have a referent in the text. For footnotes, Pandoc uses [^x], with X the number of the footnote. Pandoc places this both at the position in the text and just in front of the note (at the end of the file). For endnotes, Pandoc uses [x^], so that we can quickly tell it apart from footnotes. Such formatting is useful since we can then manipulate it with a programming language and perhaps later even restore it to a .docx format.

Python is a programming language, and python-docx is a library you can download and run from within Python (more on Python in Chapter Seven). In this example, I used the most basic command to extract all text, and apparently this does not preserve formatting, not even the text of the notes. In this example, then, it does not particularly shine, but unlike the other options, it has tools to access the specific parts of the document: for example, if we only wanted to extract the English introduction, we could have written instructions for Python to do just that.

2 Encoding of Text

We previously established that text, broadly defined as human-readable characters in a wide sense, plays a particular role in the digital world, treated strictly differently from all other data. But if the computer can only handle zeros and ones, how is the text itself defined? Let us discuss this from the ground up.

A bit is the smallest unit stored on a device, and it can merely store as 'yes' or 'no,' 'on' or 'off,' 'zero' or 'one'. The next smallest unit is a group of eight bits called a byte. Since each bit inside this group can be 0 or 1, there are 256 (2^8) combinations. There is no need to make each combination represent a number

7 I simply used the command: pandoc -f docx -o pandoc.txt Original.docx

because the numbers zero to nine are sufficient to express any number in our decimal system; with multiple bytes, we can encode as large a number as we like. The other 246 slots are, therefore, available for other things such as the alphabet and punctuation. Imagine these combinations of zeros and ones written out underneath each other, with 00000000 at the beginning and 11111111 at the end. If such a list has a character next to it, we have a simple way for a computer to look up what to print to the screen when encountering a specific combination of zeros and ones. An early lookup table that gained tremendous popularity and is, therefore, still shaping modern encoding is ASCII (American Standard Code for Information Interchange). ASCII only defines a list of 128 elements; in other words, it can suffice with 7 bits.[8] In ASCII, 1000001 is 'A' and 1100001 is 'a.' ASCII has some basic punctuation and extra characters, totaling 95. If you have an American keyboard, you will notice that all of them have survived till this day: 47 double-labeled keys plus the spacebar. The other combinations of zeros and ones define the so-called control characters. Some of them can also directly be typed, such as Tab, Return, Escape, and Backspace. ASCII defines a sorely insufficient list, as characters such as é, ç, ß or ø are not in it and, therefore, cannot be encoded despite their prevalence in European languages. Scripts other than Latin are simply not available. It is, however, still good to know of this heritage, since there is a small chance you will encounter ancient software that only recognizes ASCII. By giving it differently encoded text, in other words, that uses a different lookup table, the software will output something completely wrong. In my own work with Arabic, this often happens in a non-obvious way, where the Arabic is displayed correctly in the PDF but once selected, copied, and pasted, it shows garbled characters.[9]

The one encoding you need to know of is UTF-8, short for 8 bit Unicode Transformation Format, often simply referred to as Unicode.[10] Most companies and organizations in the tech industry support or recommend Unicode, which has made it the de facto standard. By using multiple sets of eight bits, UTF-8 can define up to 1,114,112 characters, of which so far 137,374 have been assigned. A thorough discussion of Unicode is outside the scope of this book, but it is worth noting that more and more ancient scripts are supported in Unicode. This is a major boon for us, as we can digitally transcribe manuscript sources in a digitally-native way without any hacks or tricks. Nonetheless, it

8 This is a remnant of a computer era in which the standard of 8 bits as a byte was not fixed yet.

9 A technical term for this phenomenon is mojibake, a term with Japanese origin.

10 In the rare occasion that a program is not reading your text file correctly, you might want to look into UTF-16 and UTF-32.

remains good to acquaint yourself with the Unicode support of your particular script, to understand its history, benefits, and drawbacks. As an example, take the following excerpt:[11]

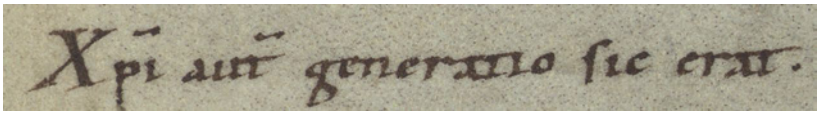

Xpĩ aut̃ generatio ſic erat.

FIGURE 5.1 Encoding a manuscript

This represents five words from the Gospel according to Matthew, Chapter 1, Verse 18. We could simply transcribe it as "Christi autem generatio sic erat." If we want to be more faithful to orthography, we could also transcribe this as "Xpĩ aut̃ generatio ſic erat," where we use one Unicode character for the ĩ, namely "Latin small letter i with tilde" (U+0129) and two Unicode characters for t̃, namely "Latin Small Letter T" (U+0074) and "Combining Tilde" (U+0303).[12] Computers that decode Unicode text will understand 'combining characters' as a character that should not receive its own space but should be placed above or below the previous character. Visually on screen, the result is one character, t̃. We further changed the s in 'sic' for ſ, which is the "Latin small letter long s" (U+017F). You may have noticed the transcription of 'p' as the second letter of the first word. This word was simply understood as consisting of the Latin letters X, p, and i, forming a *nomen sacrum*. If we, for one reason or another, wish to note its Greek origin as Chi, Rho, Iota, we could encode it like this: Χρι. Notice the use of "Greek Capital Letter Chi" (U+03A7) and not "Latin Capital X" (U+0058), even though visually they are indistinguishable. We can see the flexibility of Unicode: without installing any additional software, I can easily type out these different transcriptions, and I am assured that they are understood by most computers.

However, there are caveats. First, "Χρĩ aut̃" may at first sight appear to be a very accurate digital rendering of the manuscript evidence, but the digital

11 "Reichenau Gospels." MS W. 7, Walters Art Museum, Baltimore, 11th c., f. 17r. The digital image is 5137 x 5948, showing one page at about 91,7MB. The cut is good and color balance excellent.

12 This notation, capital U, the plus sign, and a combination of numbers and letters, is the standard way to refer to a specific Unicode character. This combination can be used to directly write the character. On Mac: Install the Unicode Hex Input keyboard, select it, and hold down *Option* while typing the combination. Windows: Hold down *Alt*, press + followed by the combination. Linux: Press *Ctrl+Shift+u*, followed by the combination.

tilde first goes up and then goes down, whereas the macron in the manuscript first goes down and then goes up, and there is nothing one can do against this. Second, even though the different ways of encoding are easy for people to understand, it can cause issues when a computer performs a search. If I transcribe the text as Xrĩ auι̃, a software that is not programmed flexibly enough might read the second word as being of four characters in length. It would then give a wrong character count. Or what if we search for “Χρι” and the software sees ‘ι’ as something different than ‘ι̃’? After all, U+03B9 is only one character while U+03B9 combined with U+0303 is two. Such cases show the importance of normalization: when a computer needs to manipulate the text, it should read ‘i,’ ‘ĩ,’ and ‘ι̃’ as the same thing. This kind of problem can occur more often when you work with non-Latin scripts. It becomes more frequent when the script becomes more obscure. This is because the available software does not significantly take into account such scripts. Here is an opportunity for us to take responsibility ourselves and work out issues as best as we can on our own. To do this, we need to operate on a level that allows enough flexibility—and by this, I mean we need to, if necessary, be able to parse text to simple text file formats such as .txt or .xml and be able to write code that takes the text apart in whatever way necessary.

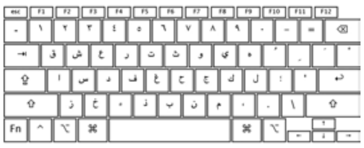
There is one more thing to discuss with regard to text encoding, and that is fonts. Unicode’s lookup table, which consists of more than 100k characters, is still an abstract list. A computer can only display it on your screen by using a font. A font is again a lookup table, but this time with Unicode on the left and a glyph on the right. Since fonts are restricted to 65,535 characters, no font can contain all of Unicode’s characters.[13] It is possible, then, that a computer first looks up which Unicode character belongs to the binary encoding, then looks up which glyph belongs to that character, and finds nothing. An empty box (sometimes described as a slab of tofu) will appear, or a black square with a question mark. This can be an issue when sharing documents. For example, when you write something in Word, the computer will ‘remember’ not only which characters you have typed, but also in which font. Somebody else will open that document on another computer, and the document will instruct the computer to render the characters in that font. If the font is not installed on the other computer, it will switch to a different font. If that different font does not contain the glyphs needed to render the encoded text, you are going to have a problem. This problem does not exist when exporting to PDF since you

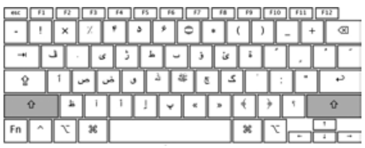
13 OpenType and TrueType, the most common font formats, do allow for one file to combine different font files, thereby being able to go over this restriction.

can instruct in the settings that all fonts used are to be included—in other words, the font you use is included in the PDF you make, which allows anyone else to see the document as intended.

There are two technologies that can make your life easier when it comes to entering digital characters. The first is to create your own keyboard layout. Consider your keyboard: every key has got something printed on it, like a letter Q or the number 5 and a % symbol, or the command Caps Lock. In the US, the standard keyboard layout is QWERTY, while in France they have AZERTY keyboards. A Norwegian keyboard has the letter Ø and Æ printed on the keys right next to L. In short, we know that it is a matter of choice that a Q or Z appears when we hit the key on the top-left next to Tab, or, in other words, that either U+0071 or U+007A is transmitted to the computer. We can, therefore, just as well create our own layout, where we stipulate exactly what Unicode character (or even combination of characters) is fed to the computer when we press that key. For Arabic, this has been an essential technology to speed up my typing, since the standard layouts for Arabic have the characters under the keys that have no relation to the letters on a standard US keyboard. My starting principles were to be able to type Arabic as though I was transliterating using the standard QWERTY layout and to have a layout that has everything on board so that I do not have to move my hands away from the keyboard and use the mouse to click through menus. Variations on a letter, such as emphatic letters, letters with hamza, letters particular to Persian, and letters without *iʿjām* are accessed by pressing Shift plus the closest Latin letter. A full set of vowels is included, as are a few ways to mark the beginning and end of a quote or a Koran verse. I even included an option to type something in Latin alphabet by holding down *Option* (on macOS). To make this, I used the application *Ukelele*, which has an intuitive GUI. This allows me to type Arabic smoothly system-wide no matter the application I am using.

After creating a custom keyboard layout for Arabic, I turned my attention to the other major challenge involved when writing about Islamic history: to transliterate Arabic in Latin. I used a slightly more advanced feature of Ukelele that allows you to enter a modus with an alternative keyboard layout only





FIGURES 5.2A AND 5.2B Keyboard layout for Arabic without and with Shift pressed

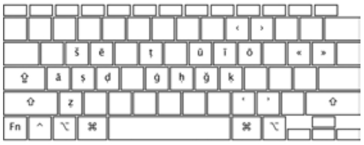


FIGURE 5.3
Keyboard layout for Arabic transliteration

when you press a specific key combination. I programmed three such combinations: *Option* + ., *Option* + *Tab*, and *Option* + \. The benefit of setting up three combinations is that if one combination is taken by an application, you can still use another one. This layout has replaced my normal layout, enabling me to type letters such as č, ē, ō, or ḍ.

Keyboard layouts are great for systemwide access to often-used characters. But if you need a rare character often for a particular project, or full names or titles that you keep using again and again, another highly practical technology that you can make your own is to run a text expansion application. This is a small program that runs in the background, where you can configure what you want to appear when you type only certain character combinations. This can be quite advanced with dynamic elements such as dates and markup such as italics, bold, or new lines. I have found it useful to set capital initials of names and titles to expand into their entirety. For example, "IA" is set to expand into Ibn ʿArabī and "NDT" expands into Naṣīr al-Dīn Ṭūsī. Since this happens at the very moment you finish typing the last initial, it feels natural, and you keep typing, only now you progress much faster since you do not have to type the unusual transliteration characters one by one.

3 Markup of Text

The previous example, where we considered a manuscript witness of "Christi autem," naturally raises the question: how can we avoid having to choose to encode either the textual, orthographic, or paleographic content and instead encode them together? For example, a user might be searching for "Christ," and we would want "Xϱĩ" to show up as a hit. This, in turn, raises another question: how can we add other remarks, such as about grammar, semantics, topics, persons, and dates? Connecting transcriptions of different manuscript copies of the same text, noting variants of the same word or passage, is yet another way to ask this same question. The ability to register all these aspects digitally depends highly on your workflow.

If you work within a word processor, the possibilities are limited.[14] You now basically work straight from manuscript to published form, which can take not much more than the form of a printed critical edition in the style we are used to: a body of a text with a critical apparatus as footnotes. It is my impression that we in the humanities handle our standard word processor too often and too quick. Such word processors are alright for styling text documents and making them ready for basic printing needs,[15] but they are especially unsuitable for two common cases: note-taking and manuscript editing. Both these activities ideally operate on a level before publishing. To understand this better, we need to define the different stages of a digital workflow.

Andrews and Robinson, scholars with experience in digital editing, suggest the following stages:[16] Transcription → Collation → Analysis → Edition → Publication.

This differentiation is a good first step. The power of it lies in revealing the nature of the different parts of our work: with transcription, we work closely with the source; when we collate and analyze, we work with the digital encoding; and when we edit and publish, we work toward molding that encoding in an intelligent shape or form. Their separation of collation from the analysis is likely because of their preference for computational collation using software borrowed from Biology that supports phylogenetic analysis.[17] Their separation of edition and publication can be explained out of their preference for digital editions, from which they can suggest that one edition can spawn different publications (for example, one for the general public and one for scholars).

Apollon, Bélisle and Régnier, the editors of *Digital Critical Editions*, propose the following workflow:[18] Content gathering and preprocessing (including collation) → encoding for internal representations → transformation through algorithms → output

14 Such as Microsoft Word for Windows, Pages for macOS, or LibreOffice for Linux.

15 I am speaking here of private printing. Publishers might be willing to receive a Word file, but do not use them to do a print run for a publication.

16 Andrews, T. "Digital Techniques for Critical Edition." pp. 175–195 in *Armenian Philology in the Modern Era: From Manuscript to Digital Text*, edited by V. Calzolari and M.E. Stone. Leiden: Brill, 2014, p. 176. Cf. Andrews, T. "The Third Way: Philology and Critical Edition in the Digital Age." pp. 61–76 in *Variants* 10 (2013).

17 For example, take a look at Dekker, R.H., D. van Hulle, G. Middell, V. Neyt, and J.J. van Zundert. "Computer-Supported Collation of Modern Manuscripts: CollateX and the Beckett Digital Manuscript Project." pp. 452–470 in *Literary and Linguistic Computing* 30, no. 3 (2015).

18 Apollon, D., C. Bélisle, and P. Régnier. "Introduction: As Texts Become Digital." pp. 1–34 in *Digital Critical Editions*, Urbana: University of Illinois Press, 2014, p. 4.

They seem to base their workflow on a methodology where you first establish a stemma before you even begin transcribing.[19] They further make the last phases sound a bit enigmatic, saying nothing about what these algorithms are supposed to be and refraining from the word 'edition' and 'publication.' This emphasizes that we do not need to get stuck on creating editions, but that we can use the encoded text for whatever purpose we see fit (such as a distant reading analysis). Compared to Andrews and Robinson, we can see an important conceptual difference: Andrews and Robinson want us to come at the sources without prejudice as to encode the manuscript evidence as raw as possible. In contrast, Apollon et al. want us to formulate our goals beforehand, so that we may include the aspects relevant to those goals directly in the encoding when we inspect and transcribe the manuscript evidence.

Lastly, we make a note of the model of Rehbein and Fritze, derived from their workshop on digital editing:[20] Data modeling → Transcription and Encoding → Publishing.

The advantage of this workflow is that it does not derive from the classical method of editing but considers the workflow anew from a digital point of view, most notable from the introduction of the term 'modeling.' As the two explain, this first step can be seen as a way to outline the entire project, as the model does not only follow out of the source material but also from the desired final product. For Rehbein and Fritze, this product is clear: a critical edition. How to move from encoding to publishing is left undiscussed, perhaps suggesting that the encoded text itself is an edition.

From their workflows, combined with the arguments made in this book, I propose the following synthesis:

Digitizing → Transcribing ⇄ Analyzing → Publishing.

With this workflow, I want to foreground the photographic aspect of our work. The previous workflows focus heavily on the text as an abstract entity, thereby building in a blind spot for the material aspect of our work, and the photographs with which we work as part of that materiality. The digitization itself is best left to libraries and museums, but in our handling of these files, we ought to keep in mind the conceptual framework that I laid out in the first few chapters.

19 There are cases in which this is a necessary decision, for example, when the manuscript evidence consists of hundreds of witnesses you need a stemma to find out which manuscripts are most useful to include in your edition.

20 Rehbein, M., and Chr. Fritze. "Hands-On Teaching Digital Humanities." pp. 47–78 in *Digital Humanities Pedagogy: Practices, Principles and Politics*. Cambridge: Open Book Publishers, 2012.

I also want to bring out the interdependency of transcription and analysis, in which, I maintain, small samples of transcription ('test drills') can inform analysis such as coming to a good stemma and finding interesting aspects of the manuscripts and the text they contain. These findings can, in turn, influence the transcription and the decision regarding those aspects that require tagging during transcription. This workflow, it should be understood, also supports analysis of the photos and not just of the digital text extracted from them.

In agreement with Apollon et al., I see the step toward sharing our findings as a separate, final step. In this stage, we do not alter the encoded data we have but prepare a polished, publishable digital document based on the data and on the analysis thereof. It should be noted, however, that the data itself is also an outcome of the project and should be seen as a deliverable. This way, the workflow has a circular aspect to it: the end result of one project can be the grazing ground of the next.

Where would a word processor like Microsoft Word fit in? Working in Word would mean that we use it for basically all steps beyond digitization. Obviously, Word is not meant to carry such a burden, and as such, most of the analytical and editorial work only goes on in our head, and Word becomes a crude sketch pad out of which we slowly and dully draw our final product, which must be a printed edition since a word processor does not support much beyond that. Importantly, for everything that we use Word, we are irreparably altering the data, making it difficult to use the encoded transcription for anything else. For small projects that are solely geared toward a printed critical edition and deal with a text that almost certainly will not be of use for other bigger projects, a word processor may be the right tool. For anything more, we would do well to harness the digital power that a computer can offer when we serve it a digital and tagged text. This means that we should strive, if only in theory, to disentangle the digital data we make and save on our own computers and the digital documents (among them most importantly actual publications) we share with the rest of the world.

Perhaps on the other side of the spectrum stands the use of a graph database in which each word is a node, connected with another word that 'comes after' as an edge, also connecting to words in other manuscripts either by 'same,' 'equivalent,' 'variant,' or 'missing' edges, having attributes such as 'personal name' and 'abbreviation.' Data held together like this becomes impossible for us to read or even have a grasp on, but computers will glide through it in ways and at speeds impossible to achieve for human beings if they had a critical edition of the text under investigation.

Most people have settled on a middle ground solution, using plain text and adding special symbols that we can tell a computer to interpret in a certain way. Let us go over three different ways of doing this.

One simple way is to separate transcription from markup, as is done in TextFabric, a piece of software originally designed to analyze the Hebrew Bible but increasingly used to analyze other ancient corpora, such as cuneiform tablets. TextFabric uses no more than a collection of .txt files where a new line indicates a new bit of the text. So far, TextFabric has been used on the word level, meaning that the transcription is saved with every word on a new line, after which new text files can be created with an equal number of lines and information on the same line number as the word it informs. Thus, the simple fact of corresponding line numbers connects the markup with the transcription. For example, a file can be created on every line with the grammatical function of the corresponding word in the transcript. Once those files are created, TextFabric allows for advanced text analysis. As such, this approach is especially suited if you want to analyze a text deeply and can afford to invest time in preparing it and its markup in such a way.

Another approach is to have very light markup right there within the transcription. For example, the OpenITI initiative has developed mARkdown to structure an ever-expanding corpus of digital texts of the Islamic literary heritage, which runs over a billion words with even individual works comprising of several million words.[21] With text files that heavy, a very lightweight way of tagging is necessary. For example, one such text is a biographical dictionary with over 30,000 biographies.[22] If we need every biography tagged, then the length of a tag matters a great deal since it will be added 30,000 times to the file. This is what mARkdown proposes to do. It allows tagging headings, paragraphs and page numbers, poetry verses, dictionary elements, biography elements, names, years, and a few more other things. The tags usually start with three hashes. For example, the start of a biography of a man is indicated by ### $ and a woman is indicated by including ### $$. The end is simply inferred from the occurrence of a tag indicating the beginning of a new entry. Because mARkdown

21 The corpus consists mostly of texts that were encoded by the *al-Maktaba al-shamela* initiative, which does not employ OCR but relies on two volunteers to type out the text (so-called double keying) with an automated comparison of the two encoded texts to weed out typos. In the name of mARkdown, the A and R are capitalized to note its use for Arabic. The name itself is derived from 'Markdown', a popular lightweight syntax to give plain text some formatting, which is used by the tech industry at large, worldwide.

22 Romanov, M.G. "Observations of a Medieval Quantitative Historian?" pp. 462–496 in *Der Islam* 94, no. 2 (2017).

is project-specific, it also includes impurities such as tags that do not support research goals but are included to avoid malfunctioning of the particular technology the project is using.[23]

A middle ground between heavy and light markup is the third way we will look at, and this is by far the most popular one within the humanities. This third way is to use XML, which stands for eXtensible Markup Language. Plain text files that adhere to the rules of XML can be saved as a .xml-file. We will look at XML more closely in the next chapter. Let us here suffice with the two leading principles of XML. (1) An .xml-file consists of elements. Elements are easy to identify since they have a tag to mark the beginning and one for the end.[24] The beginning-tag looks like <N> with N a word of your choice, and the end-tag looks like </N> with N being the same word as before. (2) In a .xml-file, all elements are properly closed, and none overlap. If you consider these two principles, you may notice that it is not saying much. It is only declaring some sort of meta-structure, but it says nothing about how to tag a name or an abbreviation. The choice is left to yourself. Fortunately, these issues have been thought through before and the most significant result of that has been achieved by the TEI, short for Text Encoding Initiative. The "TEI Guidelines for Electronic Text Encoding and Interchange" (or simply: TEI) have been developed specifically for humanities research and give recommendations for how to call your tags and what and how to use it. Given the breadth of humanities research, TEI has ballooned to a size that can be, at first, overwhelming. However, projects seldom need more than a small subset of TEI, which makes it easier to familiarize yourself with the rules and tags that TEI offers. Even so, given that the same phenomenon can be encoded in multiple ways with TEI, the guideline itself requires a guideline to its best practices, preferably in the form of a workshop by a veteran TEI encoder.

On the following page, we see our example text encoded in TEI.

23 Says Romanov: "While EditPad Pro [the particular technology the project relies on, CvL] handles large files very well, it has problems with long paragraphs (or, more correctly, lines). For this reason, long paragraphs are split into shorter lines, where each line starts with ~~ (two tildas [sic])."

24 There is also an exception where beginning and end tags are combined into one.

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>One Line from the New Testament, a Digital
Encoding</title>
        <principal>L.W. Cornelis van Lit o.p.</principal>
      </titleStmt>
      <extent>less than one Bible verse</extent>
      <publicationStmt>
        <authority>L.W. Cornelis van Lit o.p.</authority>
        <date>2020</date>
        <idno type="DOI">10.1163/9789004400351_007</idno>
        <availability>
          <licence target="https://creativecommons.org/
licenses/by/4.0/">
            <p>CC BY 4.0 applies.</p>
          </licence>
        </availability>
      </publicationStmt>
      <sourceDesc>
        <msDesc>
          <msIdentifier>
            <settlement>Baltimore, MD</settlement>
            <institution>The Walters Art Museum</
institution>
            <idno>W.7</idno>
          </msIdentifier>
          <msContents>
            <p>This line comes from the Gospel according to
Matthew, Chapter 1, Verse 18.</p>
          </msContents>
          <physDesc source="http://thedigitalwalters.org/
Data/WaltersManuscripts/ManuscriptDescriptions/W7_tei.xml">
            <p>This encoding was based on a digital
manuscript. The highest quality available gave an image of
5137 × 5948 pixels at 91.7MB, showing one page, the color
balance is excellent as is the cut. See @source for link
to original TEI file with codicological information of the
physical manuscript.</p>
          </physDesc>
          <history>
            <origin>
              <p>Probably from Reichenau Abbey, Germany,
11th century.</p>
            </origin>
          </history>
          <additional>
            <surrogates>
              <bibl facs="https://iiif.archivelab.org/iiif/
W7000037600/514,3353,1270,166/full/0/default.jpg">
                Image of only this line.
              </bibl>
            </surrogates>
          </additional>
        </msDesc>
      </sourceDesc>
    </fileDesc>
    <encodingDesc>
      <projectDesc>
        <p>This encoding is an example for the book 'Among
Digitized Manuscripts' published by Brill, 2020.</p>
      </projectDesc>
    </encodingDesc>
  </teiHeader>
  <text>
    <body>
      <p>
        <pb n="17r" />
        <lb n="15" />
        <name type="person">
          <choice>
            <orig>
              <choice>
                <abbr>
                  <foreign xml:lang="grc">Xρῖ </foreign>
                </abbr>
                <expan>
                  <foreign xml:lang="grc">Xρι</foreign>sti
                </expan>
              </choice>
            </orig>
            <reg>Christi </reg>
          </choice>
        </name>
        <choice>
          <abbr>aut </abbr>
          <expan>autem </expan>
        </choice>
        generatio
        <choice>
          <orig>ſic </orig>
          <reg>sic </reg>
        </choice>
        erat.
      </p>
    </body>
  </text>
</TEI>
```

The shape and color of this example are typical of XML: every tag has its own level, and this makes triangle shapes when there are tags within it ('children'). Tags, attributes, attribute values, and tag values all have different colors.

What may be surprising is how much space is allocated to things not directly related to the marking up of the text. Whereas in TextFabric the markup and text were separated, and whereas in mARkdown the markup could be said to be within the text, TEI allows us such an elaborate markup that it is starting to look like the other way around: the text is within the markup. When you look closely, you will notice that the document is made up of a teiHeader tag and a text tag. This is actually one of the strengths of TEI, in that it leads to a self-contained file. Given the ability of digital documents to be copied and transmitted freely, the context of this file might get lost. With the information in the <teiHeader>, someone who stumbles upon it will be able to figure out what it is, and what they can do with it.

The header starts with a file description, which, in turn, starts with a title statement, meaning the title of this digital document. In between the title statement and publication statement, I included an extent tag. A future reader, I can imagine, would open this file and worry that it is missing most of the text since it contains only a single line of marked-up text. With this tag, I can reassure those readers that that is correct. The publication statement then gives legal information, after which a source description takes up the remainder of the file description. In this source description, I identify the manuscript. Since we are working from a digital photo, I make this clear in the following physical description. Admittedly, my use is stretching the definition of the physDesc tag, since the TEI guidelines would actually want you to consider and describe the physical manuscript. The guidelines were written, then, with the use case in mind, where you have direct access to the manuscripts (or the authors simply think of digitized manuscripts as neutral windows onto the physical artifact, as described and argued against in Chapter Two). TEI only allows you to define digital surrogates in a separate tag, for which I have used an IIIF reference. This is again not exactly how it is supposed to be used, but currently, there is no good alternative. We shall discuss IIIF later on in this chapter. After this long description of the file, I included a snippet about encoding to bring to attention that this document is an example belonging to this book, and to explain the context of creation and the purpose of this file.

When we finally come to the text, I wrapped everything in a body and a paragraph tag as is customary. I included a page and line number to signal the larger context of the sentence encoded here. The first word is immediately the most complicated to encode. Certainly, there are other ways of doing it, and perhaps one way is preferable over the other for reasons of performance, such as the time it takes for a computer to parse search queries. In our case, I opted in the first place to mark up the word as a personal name. This way, if the document is considerably larger, we could do a query on the personal names to find out, for example, how many times a name is used or how many unique names are used. You could install a more complicated system, including an ID that links to an entry in a list that gives specific information about that person—something I did not deem necessary for this example. Second, I encoded two choices: one form of the word with Greek characters, and another, normalized form of simple Latin characters. Further, for the first form, I provided two choices again: either an abbreviated or unabbreviated form. Lastly, I emphasize the foreign letters by including a foreign tag. For the next word, I did something similar to encode both the abbreviated form and the expanded form. For *sic*, it is the same, this time choosing between the original and the modern manner of writing.

This is as far as our discussion of TEI will go in this book. As with every tool and technology, for some situations, it will perform excellently and in another, it will not. What is especially troubling is that TEI can only be used for texts that are an 'ordered hierarchy of content objects,' meaning that its dissection needs no overlapping of tags. This restriction flows from TEI being a flavor of XML. It assumes that texts can (should?) be linearly ordered, and this excludes a lot of humanities research.[25] For example, in cultures with extensive commentary traditions, a text can be construed from an almost organic process of text reuse. To properly edit and analyze such a text, these other commentaries and base works need to be taken into account. For this, overlapping tagging would be necessary. Post-classical Islamic intellectual history is such a field. Islamic studies, in general, encounters another difficulty, namely the awkwardness of mixing an Arabic source text and its right-to-left script with English tags with its left-to-right script. Even if we disregard these problems, it still depends on the savviness and experience of the user to unlock the benefits of TEI. XML was originally conceived to be a form of markup that leverages the best of both worlds: discursive enough for human beings to read it, regular enough for machines to interpret. But the extensive apparatus that TEI prescribes makes it difficult for people to simply read and understand what is going on. We would need an interpreter, a piece of software that converts all the tags in visual markup, such as making titles bigger and bold and showing personal names in italics. This not only counts for reading but also writing. Writing TEI tags from memory is too difficult. However, solutions that provide automatic tag suggestion as soon as you type the first letters are labor-intensive,[26] and those that provide a set of buttons you need to click with a mouse to add a tag are ergonomically frustrating. This begs the question: if most of what we mark up by hand might be performed automatically in just a few years—when algorithms to detect names and dates are robust enough, for example—if so, is it not an absolute waste of resources to type all of it by hand? "Almost anything we do now, with painstaking effort, may be automated in five years' time,"[27] says musicologist Julia Craig-McFeely. This snarky remark should be kept in mind at every step of the way while marking up a text, especially when the choice has been made for TEI.

25 The TEI Guidelines itself offers some solutions but these either cause bloat, inconsistency, a lot of extra labor, or all of the above.

26 Such as in <*oXygen/*>, a favorite XML editor among Digital Humanists.

27 Craig-McFeely, J. "Finding What You Need, and Knowing What You Can Find: Digital Tools for Palaeographers in Musicology and Beyond." pp. 307–39 in *Kodikologie Und Paläographie Im Digitalen Zeitalter 2*, edited by F. Fischer, Chr. Fritze, and G. Vogeler. Norderstedt: BoD, 2010, p. 330.

4 Intermezzo: Using the Right Editing Tool

If Microsoft Word (Windows) or Pages (macOS) or LibreOffice (Linux) are not the right environments to do our philological work, what is? The previous section already hinted at the correct but frustrating answer: it depends. And it depends not only on the research questions and the desired outcome but also at the moment in time since software development is constantly in motion. Several distinctly different use cases are discernible, I think, that will hold up regardless of the future technological development.

4.1 *You Need to Create a Small Critical Edition Solely for Print Purposes*

If there are no unusual features to the manuscript evidence you are working with, and all you need is to create a print publication, you might have enough with software such as Mellel or Classical Text Editor.

Mellel (macOS only) is a good example of an extended word processor.[28] It behaves similar to Microsoft Word but has several improvements that will facilitate scholarly writing. Above all, it handles large documents with significantly less trouble. Entire books or dissertations can be written on it as one document with virtually no lag. The find and replace functionality in Mellel is very advanced; it supports regular expressions and the option to save search queries. Cross-references throughout the document pose no problem and do not transform into a burden as the document grows. The same goes for footnotes, of which Mellel offers the ability to offer many different note streams, i.e., different sets of notes. For example, you can create a different series of notes for critical apparatus and a different one for comments. These streams can be in different scripts: Mellel has sophisticated support for blending scripts and writing directions. It is often the little things that push one piece of software far ahead of others. For example, with Mellel you can not only underline text (in more than a few ways, in fact), but you can also overline text. This is actually what is done in Arabic, so to be able to replicate that in an edition greatly improves the aesthetic experience.

The Classical Text Editor (Windows only) comes into play when a little more power is needed. It should be noted from the outset that its price-tag matches its learning curve: high. Since adoption is low, it is hard to get good training

28 For my everyday writing I use Nisus Writer Pro, which supports Arabic, handles large documents without a problem, and has several useful extras such as a mode in which text is white on a black background, everything but the text is invisible, and the line on which you are typing remains in the middle of the screen, which is excellent to work without distractions.

material. Notwithstanding, it is, to date, the most complete and most advanced attempt at providing an ergonomic critical editing environment. CTE allows you to encode and markup the edited text, its variants, notes, translations, and anything else, within a graphical user interface without having to code or laboriously have to piece together the different bits of information. It does so by inviting you to store the different bits of information in different documents, while still seeing all those documents on your screen at the same time. This allows for elaborate critical apparatuses and extensive notes as you no longer feel bound by the dimensions of a piece of paper, as you do in Mellel. Technology, here, works for you in expanding your potential. This is equally true for CTE's export functions. You can transform this bundle of documents into one beautifully typeset PDF, one that rivals the looks of LaTeX, and you can also export it as a TEI-compliant XML file.

4.2 *You Are Working with a Very Large Text from Which You Will Mine Specific Bits of Information*

In this case, you should pay special attention as to how you will share your final data and what you want to publish out of it. In a nutshell, you could draw out your workflow, starting with the end, and take a few steps back—from the publishing phase to the analysis phase to the transcription phase—to see what technology you will use and how you can make sure that you have the easiest time using it. 'Mining' could be as easy as performing a search through the document for specific tags and then visually evaluating if the passage is relevant for your research or not. It could be as advanced as performing automated analysis using a programming language such as Python, where you will only look at the final statistics that it gives back. In either case, an approach like mARkdown is good. For this you can use a code editor like Visual Studio Code, Sublime, or OpenITI's favorite, EditPad Pro. Since the document will be too long and it will be difficult to go over it twice, you will have to be very sure what to tag so as to catch everything you need on the first run itself.

4.3 *You Have a Large Set of Separate Writings*

If you have something akin to a diary, collected letters or poems, or a journal or log, you will find much use in a VRE like Scripto. This will allow you to keep an overview of what has been transcribed and what has not, and you will be able (but not obliged) to recruit others to help you. You might simply transcribe the text without markup, or you could adopt a lightweight markup like Markdown, by which I mean the regular industry standard, not the specialized Islamic studies one. The advantage of a lightweight but broadly supported markup language like Markdown is that you will have a very easy time converting it to a webpage or printable document, as there are many applications that will recognize it and convert it to HTML.

4.4 *You Will Work Intensively with a Stable Text*

If your analysis is not so much concerned with variants across manuscripts copies but more about the structure and content of the text, and if the publication of the text itself is not interesting, it will be beneficial to use a stand-off annotation as implemented by TextFabric. In this case, you will work mostly in a code editor of your choice, using regular expressions to mold the text in the shape you want it to be. For example, you may obtain or produce a full transcript in plain text without any markup, replacing all the spaces with 'new line' characters to place every word (or, word group) on a separate line. If you are still working on your programming skills, you will rely on software like TextFabric, so it is crucial to abide by the required format of the software you use. Texts falling in this category are typically already published, but a heavily annotated or thoroughly reworked version might still be publishable, perhaps not as a print publication but as a digital data set. In this workflow, the information is maximally divided into different puzzle pieces, so it will take more effort to combine all these pieces together to get a printable document. The upside is that this workflow supports the heaviest and most detailed annotation; and therefore, there are no puzzle pieces missing to convert these different files into, for example, one TEI-XML file. The most obvious tool to put these puzzle pieces together is to write a custom Python script.

4.5 *You Wish to Critically Edit a Text with Unusual Features or Multiple Manuscript Witnesses*

If you have multiple manuscripts of an unedited text, or manuscripts with unusual features, and your primary intent is to publish the text as a critical edition (digital or print), word processors like Mellel or CTE will significantly hold you back. You will, in the long run, gain more from writing up your editorial work in XML format, for example, by using a subset of the detailed annotation system of TEI. You can do this in a code editor, or use a professional XML Editor. oXygen is a popular choice in the humanities, as it has support for TEI, meaning that when you start typing a tag, it will auto suggest whatever is available within TEI that starts with those letters. This makes typing in TEI faster and more precise.

You stand to gain the most from working with a subset of TEI that has proven its worth within your specific field. For example, *EpiDoc* has been highly successful in transcribing epigraphic evidence, and the *Scholastic Commentaries* *and Texts Archive* has worked on implementing TEI for medieval commentary traditions. The latter is an especially interesting example, as the person behind it, Jeffrey Witt, has created an entire toolset that includes functions to archive and publish your editions (both online and as a well-formatted printable PDF). This can be used freely. If at all possible, you want to forego reinventing the wheel, and for that reason it is vital to find out what technology has already been developed in your own field, or a neighboring one. In the case of *SCTA*, there are other benefits too, to adopt or at least be compatible with their standard. For example, this makes your edition interoperable with other editions. This means that if the text you work on cites another text within the scholastic corpus, you can include a direct link to that text (and a link back will be created, too). Whereas editing remains, in my view, the domain of the lone scholar, here we encounter great benefits from collaborating on the technical parts of our work. Indeed, basing your work on the toolset surrounding SCTA means that you need only very little skill with programming languages. It minimizes your time writing code or agonizing over how you will move from XML files to a printed edition, and puts the focus on actually producing the annotated transcription. Meanwhile, you are not locked in, so if you want to take your TEI-XML in a different direction, you can still do so. For example, if the manuscript evidence shows a great variety or only one or a group of witnesses attest a markedly different text, you could use web development technology to build a custom interface to include on/off buttons for different witnesses and arrange them horizontally or vertically, in different colors, to convey this information to readers.

4.6 *Concluding Thoughts*

What most choices have in common is that you are encouraged to work in text file formats using code editors or VREs. This separates data entry and analysis from publication. If you are unfamiliar with code editors, this may sound like a tall order, but advanced analysis demands advanced tools, and by separating raw data entry from polished publication, you have more freedom to mark up what you want. In addition, working in this way will ensure that you keep all your options open if you want to take your research in a different direction. For example, you may initially think you want to produce a critical edition, but later on, you may come to think of text mining analyses you would love to do. By having your raw transcription in a plain text format with regular markup, it will be very easy to reshape your text to prepare it for automated analysis. All of this will take varying levels of technical know-how, but by adopting the existing standards and tools, and with practice, you will minimize the time needed to learn or get trained in using those technologies.

5 Handling Images

Editing the text in a manuscript requires access to the manuscript and as part of a digital workflow it will be beneficial to have access to images of the manuscript. It is best to have those images in your private possession. Since there will always be more images not in your possession, you will need to outfit your workflow to also allow for externally hosted images. Likewise, if you can, it will be best to eventually share the images you used so that colleagues have a chance of verifying your analysis, similar as to how in the sciences raw data is made available to replicate the analysis. There is, therefore, a nearly constant flow of data between your own computer and others, mostly over the internet.

Pronounced 'triple eye eff,' the International Image Interoperability Framework is a rather elaborate and technical standard for image storing, sharing, and showing. Most of this technology is of no concern to us in the humanities: we can safely leave it to librarians, curators, and other specialists to make collections digitally available through IIIF. Nevertheless, knowing how IIIF works and having a passive knowledge of it, perhaps with practical experience in some regards, will be a great asset to your toolbox. This is not only true because you will encounter the IIIF standard a lot, but also because smart use of IIIF can provide some real benefits. For our purposes, IIIF introduces two technological feats: (1) images that are stored in different places can be loaded using a similar command, meaning that they can be brought together almost effortlessly, and (2) a basic set of manipulations can be directly requested when loading an image, such as only requesting a specific area or a specific quality of the image.

IIIF achieves interoperability and responsive loading by having users not directly querying the image file as it is on the server, but instead, users send a request to an API with some parameters, among them the ID of the image. In turn, the API does not use that ID to go directly to the image file, but it first looks at an abstraction of the image, called a manifest, which is a JSON file. Let us unpack those two sentences step by step, with the help of an example, to understand the conceptual framework that IIIF provides. If we return to our example of encoding "Christi autem …" and we want to have a photo of the manuscript folio of that sentence, we could simply fill in the direct URL of the photo on the website of the Walters Art Museum in a browser to get the image at the highest resolution possible.

Entering such a URL in a browser is what it means to request the image directly. This gives us all 91.7MB of the photo, and for storing the research data on your own computer, this method works just fine. But what if I want to associate the encoded text with the specific excerpt of that folio? What if I want just that fragment because it will be a smaller file? I could open the large file, cut out the line, and save it as a new image file, but this will produce a proliferation of new files, and unless I would give them a meaningful file name, it will be unclear what they actually are if they are copied and moved around. Besides, I might simply want to refer to the file as it is originally stored by the library or museum, as that will have a better chance of being persistently available online. IIIF's Image API is here to help us. API stands for Application Programming Interface and is, in general, a way for computers to talk to each other. You give a formulaic request to the API, and you get a reliable, formulaic response. For example, if you want to know a zip code, you could browse yourself to Google Maps, enter an address, and read it from your screen. You could also write a script (e.g., in Python) to look it up for you. Obviously, a computer cannot see, so it has no use of browsing maps.google.com. Instead, what it can do is send a request to maps.googleapis.com, specifying the address and asking to get a zip code back. Similarly, for referring to the manuscript image, we want to make a consistent, reliable reference, not just to the image as a whole but to an aspect of the image.

The Walters Art Museum does not support IIIF (yet), so our example needs to take a detour. This will only benefit us, as we can learn to take matters in our own hands and make use of IIIF more actively. Using IIIF to refer to an image means, unfortunately, that we cannot simply have an image stored wherever; it needs to sit on a server that runs the Image API. For simple purposes, this is easily done by uploading the image to Archive.org.

Using the new URL will make the image appear only slowly, as again the entire 91.7MB is downloaded. Archive.org, however, offers IIIF support if we go to a different URL and use the ID of the object, which is, in this case, 'W7000037600'.[29]

We now see the image through the IIIF Image API, and the first noticeable difference is that it appears almost instantly. This is because it is first offering us a dynamically created version of the image, which is very small, and once it is loaded, it will load a higher resolution in the background. In the top-right, we see a button 'Enable Cropper.' If we use it and select the region of interest, a pop-up appears with the desired URL. This functions as a query with several parameters, as such:

29 A remnant of its origin: Walters MS W.7, f.17r, which must equal to the 37th photo, taken at 600dpi. Notice too that I provided some metadata on archive.org to describe the photo and link back to its origin.

/iiif/ID/REGION/SIZE/ROTATION/QUALITY.FORMAT

With the ID, we specify which among the many images on that server we are interested in. With the region, we can ask to offer only a cut-out. We do this by specifying the x,y-coordinates of the top-left corner and the bottom-right corner of a rectangle. Changing the size means the image you will get will have a different pixel-dimension. Rotation does just that: it will give you back (a region of) an image at an angle, which is very useful for marginal notes that are slanted or upside down. Quality can be changed to get a gray-scale or bitonal image. File format, finally, can give you an image in a format of your choice as long as the API can do the conversion. In this way, we ask the IIIF Image API to work on W70000037600 and only return us the region with the top-left point being (514, 3353) and the bottom-right point being (1270, 166), as big as the image allows, no rotation, in color, and as a JPG file (as opposed to the original, which is stored as TIFF). Notice, for example, how the resulting JPG weighs just 40KB, which is of course much easier to work with than the entire 91.7MB file.

Before the image file itself is requested, two other files are used: a manifest.json and an info.json. JSON is short for JavaScript Object Notation and is quite similar to XML, except it does not work with tags wrapped in angle brackets, but with "key: value" pairs. We will work with it in the next chapter. When you encounter an IIIF-compliant image, you can lift the curtains and look at what is going on behind the scenes, by requesting these files directly to look at their textual content. Archive.org automatically creates such a manifest file.

Let us write our own so that we get to understand the conceptual framework behind IIIF and experience using manifest files as a building block in our workflow. The one constructed for our example can be seen on the next page.

```
{
  “@context”: “http://iiif.io/api/image/2/context.json”,
  “@id”: “http://iiif.archivelab.org/iiif/W7000037600/
manifest.json”,
  “@type”: “sc:Manifest”,
  “attribution”: “The Internet Archive”,
  “description”: “Resources for Ch. 5 of <i>Among Digitized
Manuscripts.</i><div>Standards for Digital Editing or Editing
Digitally.<br /></div><div>Page and excerpt taken under CC0
license from Walters Art Museum, available at\u00a0https://
manuscripts.thewalters.org/viewer.php?id=W.7#page/1/mode/2up</
div>”,
  “label”: “Among Digitized Manuscripts Chapter 5 Standards
for Digital Editing or Editing Digitally”,
  “logo”: “https://encrypted-tbn2.gstatic.com/images?
q=tbn:ANd9GcReMN4l9cgu_qb1OwflFeyfHcjp8aUfVNSJ9ynk2IfuHwW1I4mD
Sw”,
  “metadata”: [
    {
      “label”: “title”,
      “value”: “Among Digitized Manuscripts Chapter 5
    Standards for Digital Editing or Editing Digitally”
    },
    {
      “label”: “subject”,
      “value”: “Digital Humanities”
    }
  ],
  “seeAlso”: “http://archive.org/metadata/W7000037600”,
  “sequences”: [
    {
      “@context”: “http://iiif.io/api/image/2/context.json”,
      “@id”: “http://iiif.archivelab.org/iiif/W7000037600/
canvas/default”,
      “@type”: “sc:Sequence”,
      “canvases”: [
        {
          “@id”: “http://iiif.archivelab.org/iiif/W7000037600/
canvas”,
          “@type”: “sc:Canvas”,
          "height": 5948,
          "images": [
            {
              "@context": "http://iiif.io/api/image/2/
context.json",
              "@id": "http://iiif.archivelab.org/iiif/
W7000037600/annotation",
              "@type": "oa:Annotation",
              "motivation": "sc:painting",
              "on": "http://iiif.archivelab.org/iiif/
W7000037600/annotation",
              "resource": {
                "@id": "http://iiif.archivelab.org/iiif/
W7000037600/full/full/0/default.jpg",
                "@type": "dctypes:Image",
                "format": "image/jpeg",
                "height": 5948,
                "service": {
                  "@context": "http://iiif.io/api/image/2/
context.json",
                  "@id": "http://iiif.archivelab.org/iiif/
W7000037600",
                  "profile": "https://iiif.io/api/image/2/
profiles/level2.json"
                },
                "width": 5137
              }
            }
          ],
          "label": "Among Digitized Manuscripts Chapter 5
Standards for Digital Editing or Editing Digitally",
          "width": 5137
        }
      ],
      "label": "default"
    }
  ],
  "viewingHint": "paged"
}
```

Here, we see that the entire document is wrapped in curly brackets. This is a JSON-way of indicating that all the information belongs together. In the case of an IIIF manifest file, the information belongs together because it describes one real-world artifact. The first two lines are necessarily included to alert any person or computer reading this file that it should be read as an IIIF manifest. The next lines, up to "sequences," describe the artifact in its general characteristics. ID is necessary so that, if we were to open multiple files at once, we could always distinguish this one from other manifest files. "Label" is a more human-friendly version of the ID. Inside "metadata" we see square brackets. This indicates that the value we want to associate with the key "metadata" does not consist of just one piece of knowledge, but multiple ones, each of them wrapped in curly brackets and separated by a comma. In this case, we only included two items, but ideally, a full codicological description is given here. We leave it out here so that the entire manifest file can fit on two pages and also because this description is already offered in a machine-readable format by The Walters Art Museum, namely in a TEI-compliant file called "W7_tei.xml." A reference to it is given under "seeAlso," together with some hints to read this file as a TEI-XML.

What we discussed so far is for IIIF what "teiHeader" is for a TEI document. Just as "text" comes next in a TEI file, what comes next in an IIIF manifest is a definition of the actual image content. To do this flexibly, images are not simply defined but painted onto a canvas. A canvas is an abstract rectangle, defined in IIIF by making its type "sc:Canvas," given it an ID, and defining its width and height. One manifest can include many canvases. For example, an entire book can (and usually should) be defined by one manifest in which each page is defined as a canvas. The image of the page is only defined within the canvas, painted onto it, with the ID in its "resource" equal to a URL that leads to an IIIF Image API, including its usual parameters. In most cases, we request from this Image API the full image in its best quality, without rotation, and paint it onto the canvas such that it fills up the entire canvas. It is, however, entirely possible to have the image only fill up a quarter and you can define three other images to fill up the other quarters. Since the images can come from different locations, IIIF gives you the potential to reconstruct a page if the torn pieces have been digitized by different institutions, or to reconstruct a manuscript if different folia are scattered across multiple libraries.

You will have noticed that the images are wrapped inside canvases, which in turn are wrapped in sequences. A sequence defines the order of canvases. For example, for a manuscript for which it is known that pages have been reshuffled, you could define a sequence of canvases in the order of which the artifact currently is, and a separate sequence in which you define canvases in the order in which you think the manuscript ought to have been in the past. You can even include empty canvases to represent missing pages.

With its nesting, a manifest JSON can be a bit overbearing to look at at first, but with the conceptual framework in mind, it need not be hard to read it. In our case, we have defined one sequence, with one canvas of 5137 pixels wide and 5948 pixels high, and on that canvas we have defined one image.

IIIF also allows references to text annotations for each canvas. Thus, we could have included a reference to our TEI encoding of "Christi autem…." I have thought it sufficient to include a reference to this IIIF manifest within our TEI file. After all, the TEI file is dependent on the IIIF manifest and not the other way around. At the moment of writing, there is no built-in support in TEI or IIIF for the other standard such that it truly leverages the strengths of both into a powerful combination. It seems reasonable to expect that the organizations behind each standard will work on this in the coming years. Since the combination of digital image surrogate and digital text surrogate is of paramount importance for anybody working with manuscripts, we will have to follow these developments closely.

Having reading knowledge of manifest files is a great asset. For example, if you encounter an IIIF viewer (such as Universal Viewer or Mirador) that shows a digitized book that allows for text searching, you know that the manifest file will contain links to images of all the pages as well as one or more links to plain text files containing (for example, an OCR version of) the text. You can open the manifest.json in a code editor like Visual Studio Code to extract those links. Upon opening, the file might just be sitting on one single line, but searching for "http" will quickly find you links to images. Studying those links, you can find out what extension the images preferably have, for example .jpg. If so, you can search, with the option for 'regular expressions' turned on, for "http.*?jpg". Regular expressions allow you to search more flexibly than just using a string of characters. For example, a period represents any character, and a star means as many of the foregoing. Therefore, ".*" means 'as many characters as possible,' and adding the '?jpg.' means that as few characters as possible are found between the 'http' and the 'jpg.' This, then, makes you find entire URLs for all images. By clicking Select > Select All Occurrences, you will have selected all these URLs, which you can now cut and paste in a new file. Since we know the parameter settings of the Image API, we can make sure the region and size are full, and the quality is native for all images to get the best possible image. By creating a clean list of URLs, we can then download them, either with a script, command line tool,[30] or an application with a GUI.[31] For digitized manuscripts that are not served through an IIIF Image API, your best strategy is to find a regularity in the URLs for the images of successive pages and write a script to download files for which the URL is adjusted every time according to those regularities.[32] If you cannot right-click and 'Open image in new tab' to see the URL, you can probably still find it by digging into the page resources.

6 Archiving and Publishing

We previously defined the workflow for digital editing as follows: Digitizing → Transcribing ⇄ Analyzing → Publishing. This book focusses on the first half of that workflow and the parts of the Analyzing-phase that are included in this book focus on what depends on manuscript images and not the text per se. The reason for this is that there are already plenty of resources regarding digital approaches to text collation, stemmatics, reception analysis, semantic analysis, topic modeling, how to build a who's who or a gazetteer, or other fascinating pieces of analysis that go along with philology. Most of these discussions combine analysis and publication into one, under the term 'digital edition.' This term is a tender topic in DH. Everybody is highly opinionated and passionate about what we ought to do. Before we finish this chapter with a discussion of digital editions, let us first delve into the essential step of archiving.

Archiving digital materials means storing a set of files together in a permanent state in the double sense of the word: permanent as a guarantee that the files will not change over time, and permanent as a guarantee that the files will remain available over time. The latter sense can be further split out into two components: available in the digital world can mean 'to persist on a hard drive,' and it can mean 'to be accessible to a wider audience.' This makes archiving easier said than done. Regarding the first sense of permanent, you face a conundrum since you want to continue to work on it. Archiving would mean to preserve its state while working on it would mean altering it. As for the second meaning of permanent, you certainly want to share it publicly so that others can verify your analysis, but you only want to do so to the extent that is legally

30 A quick "wget -i listOfURLs.txt" would do it.
31 For example, on macOS, you can paste those URLs in TextEdit and right-click and select Services > Download links with Folx, which gives you a graphical user interface to download multiple files.
32 With Python, you can use the package urllib to download items on the internet.

and ethically allowed. Further, if you intend to keep working on it, which is likely, it is only normal to make sure you are not oversharing so that nobody will beat you to the publication of exceptional insights that the data provides. There are currently no clear solutions to these problems.

For your private use, you should consider adopting the 3-2-1 rule: all your truly important data should be stored in three different places, on at least two different kinds of media, of which at least one is off-site and preferably offline. Next to that, you can think of your data in terms of circles of influence, with a core set of files well protected and in immediate reach imagined as a circle, and rings of ever less important files less protected and more laborious to access. Since you likely do not want to spend a lot of time and money on this, your scenario will probably be a variation of the following: a core set of files that make up your current projects sits on your computer in a folder synced with a consumer cloud storage solution.[33] Since you occasionally backup your computer, you are already compliant with the 3-2-1 rule. In fact, if your computer has an SSD hard drive, and the external hard drive for your backups is a regular one (with a spinning platter), you have no less than three different media, as cloud storage can be considered a different kind in this case. It is likely that your next ring consists of primary sources, secondary literature, and reference materials that you value highly or otherwise use frequently. They can sit on your computer, but are not synced to the cloud. Instead, you can choose to mirror them on an external hard drive (other than the one you use for backups). On this external hard drive, you may store additional sources, books, articles, and manuscripts that you would not use often. This external hard drive, in turn, could be occasionally synced with another external hard drive or a network attached storage unit on an offsite location. This way, your most valued literature also complies to the 3-2-1 rule. The other files do not, since they are not stored thrice. If you move around a lot, you may want to consider to change the cloud option for an SD card or USB stick, which you can carry with you.

With this need for different storage devices and the likelihood that you will sometimes exchange such devices with colleagues, you need to know of the standard controlling the normal operation of a storage device. This is a 'file system.' For Apple devices, the file system is called APFS. If you use Microsoft Windows, however, you will likely have your drives configured (even without you knowing or choosing) for NTFS. Linux users, meanwhile, probably have ext4 as their file system. These file systems do not mix and match. NTFS, for example, cannot be used on a Mac. That is, you will be able to read files, but not able to write files. Third-party (paid) software will help you here, but you could

33 Such as Microsoft OneDrive, Google Drive, Apple iCloud, Dropbox, and SugarSync.

also format the disk to another file system. If you intend to share data with colleagues or friends regularly, you would not want to have such restrictions. The solution is to format the device to the file system called exFAT. Most versions of Windows and macOS can handle this, and it can even count on some support on Linux. exFAT is a file system which can take care of files of practically any size with practically any file name, organized in practically any tree of folders.[34]

Public archiving is another task. Your first port of call should be your own university, but this can currently be a drawn out process, where you might not get the type of support you are looking for and, frankly, not a long term assurance of hosting. Next, you can look for what else is going on at a national level or in your field of studies. Perhaps, a digital text corpus or other kind of repository is forming, to which you can contribute your data of your critical edition. The upside of this is that this combats 'silofication,' which is the danger of various digital resources living in ignorance of each other, whereas their connection could provide a much more powerful resource. As a corollary, it could increase your exposure since more people will normally go to the central repository and will find you through it. Moreover, when this repository finds traction, it will become an acceptable point of reference to which you can refer in notes in publications. The downside is that you are obliged to follow the rules and standards of whoever is in charge. If they do not update the corpus often, or want to format the texts in a specific way that may be useful for their own needs but not for others, its normal usability is jeopardized. As long as you do not (or frankly cannot) make meaningful use of the corpus already, it is unlikely you will contribute to the repository.

What can you do on your own? An unusually good option has been born: Zenodo. Through this service, you can create a free academic repository that the developers claim is guaranteed to be saved for many years to come. Zenodo allows virtually all the things one would want in a repository: you can store the data, the metadata describing it, and create a DOI, Digital Object Identifier. In Chapters One and Two, we noticed that one pesky problem of the digital world is that it does not have a stable reference system like the print world, with its publication information and the page number. This DOI is the digital world's answer to that problem, meant to be a persistent and precise referent to a book, article, edition, or any piece of digital data. A similar initiative to refer to one and only one person is ORCiD. An account on the latter is free, and

34 To format a device using a Windows computer, connect the device and use the program *Disk Management*. On macOS it is called *Disk Utility*. For Linux, there is not a standard program with a point-and-click graphical user interface (GUI), but *GParted* is trusted and used a lot.

Zenodo is a way to get a free DOI for your data. Most publishers assign a DOI for your publication. Therefore, you can link the DOIs of your publications and repository data to your ORCiD and vice versa to ensure that people understand you created these documents and that these documents were created by you. A good repository should include as much of your data as possible, including images, transcriptions, markup, and code. Ideally, there should be some documentation within the file or in the filename to make clear what the file is and what it does. A readme file should come along with the bundle of files, explaining the characteristics and purpose of the data set as a whole. This readme file should also include some information about creation and ownership. Either within the readme or as a separate file, you should include a license that describes what other people are allowed to do with it. Most commonly you will simply take over a license from, for example, Creative Commons. Lastly, it is best practice to include more documentation describing how the data can best be used.

If you wish to be sure and store things twice, or you do not wish to use Zenodo, there are alternatives. For plain text files such as programming code, marked-up transcriptions, notes, and statistics, there are dedicated services whose benefits consist of functionality, such as version control, easy duplication and reworking, and, after that, reintegration. The most prominent example is the free service GitHub, discussed in the next chapter. Other data, such as images and other binary files, can be uploaded to The Internet Archive. The functionality it offers is very limited and is geared at dumping the final archive onto it. What you get in return is storage at a permanent URL, with one of the strongest guarantees of permanent worldwide accessibility.

Should you publish a digital edition of your digital files? You will have to answer that question yourself, but I can summarize the discussion to help you make that decision. For a broad overview, you can best look elsewhere.[35] Practical introductions are mostly in German.[36] A good practical introduction in English

35 I have enjoyed the summarizing discussion in a dissertation: Hörnschemeyer, J. “Textgenetische Prozesse in Digitalen Editionen.” PhD dissertation, Universität zu Köln, 2013.

36 Sahle, P. *Digitale Editionsformen: Zum Umgang Mit Der Überlieferung Unter Der Bedingungen Des Medienwandels*. 3 vols. Norderstedt: BoD, 2013; Jannidis, F., H. Kohle, and M. Rehbein. *Digital Humanities: Eine Einführung*. Stuttgart: J.B. Metzler, 2017; Kurz, S. *Digital Humanities: Grundlagen Und Technologien Für Die Praxis*. Wiesbaden: Springer Vieweg, 2015;

is Apollon, Bélisle and Régnier's *Digital Critical Editions*.[37] An excellent theoretical discussion, one that continues where this book stops, is Pierazzo's *Digital Scholarly Editing*.[38] The discussion surrounding digital editions can be succinctly summed up in the following four questions. (1) Now that our workflow happens in the digital world, and we have witnessed innovative ways of doing our research, can (or, should) we find new ways of publishing our results? (2) If we do so, when is our product 'digital' enough to be called a digital edition? (3) If we have made it digital enough, at what point is our product truly an 'edition'? (4) If we make truly digital editions, how can we make them count just as 'critical' as print publications?

The first question oscillates between *can* and *should*, and this is because those in DH who work on this topic are so enthusiastic of the possible innovative, technology-driven scholarly products that they go beyond the mere possibility and argue that the digital era has dawned and we would be foolish to hold ourselves back by the paradigm of the print world. Thus, Burdick et al., authors of the manifesto-like *Digital_humanities*, boldly write: "We are moving [...] to an era of scholarship based on the collaborative authoring possibilities of the 'great project',"[39] and a little later claim that matters of publishing "have moved to the forefront of the digital humanities precisely because choices of interface, interactivity, database design, markup, navigation, access, dissemination, and archiving are all part of how arguments are staged in the digital world."[40] They claim, then, that a change in scholarly output is inevitable since these aspects are ingrained in the fabric of the digital world. Similarly, Van Zundert writes: "it cannot be the primary intent of a digital scholarly edition to represent a scholarly edition that is better represented by a print publication." And he continues to argue that a digital edition should exploit "the idiosyncrasies of the digital environment."[41] Robinson, meanwhile, calls digital editing a revolution and sees the revolution, especially play out in "changing what we

37 Apollon, D., C. Bélisle, and P. Régnier, eds. *Digital Critical Editions*. Urbana: University of Illinois Press, 2014.

38 Pierazzo, E. *Digital Scholarly Editing: Theories, Models and Methods*. Farnham: Ashgate, 2015. My references are to the online, open access version.

39 Burdick, A., J. Drucker, P. Lunenfeld, T. Presner, and J. Schnapp. *Digital_humanities*. Cambridge Mass.: The MIT Press, 2012, p. 83.

40 Burdick et al, p. 84.

41 Zundert, J.J. van. "By Way of Conclusion: Truly Scholarly, Digital, and Innovative Editions?" pp. 335–346 in *Analysis of Ancient and Medieval Texts and Manuscripts: Digital Approaches*, edited by T. Andrews and C. Macé. Turnhout: Brepols, 2014, p. 338.

make."[42] He reflects on what a critical edition is and says: "all this proclaims 'I am a hypertext: invent a dynamic device to show me.' The computer is exactly this dynamic device."[43]

The previous question already hinted at the high standards these authors want to set for such editions. Robinson spurns features such as browsing and searching through a text: "you can do these things with print editions too."[44] For a digital edition to be digital, Robinson seems to want a) all manuscript witnesses transcribed individually, b) all transcriptions to be in both original shape and regularized, c) all manuscripts available as digital photos, and d) the ability to change or add to the edition by multiple people, preferably anyone. Van Zundert argues for a nearly identical list: a digital edition ought to have a) any extant data including all transcriptions and facsimile images and any other notes, b) additional media such as sound and video, c) full-text search, and d) the ability to change or add to the edition in a collaborative fashion, possibly open to anyone. From the previous quote of the authors of *Digital_ humanities,* we can infer they would agree with such a list.

Do we still end up with a critical edition if we would follow up these demands? Let us structure the answer to this question as counterarguments. First, to wish for full transcriptions and markup seems to wish for an objective record of the facts. Next to the question whether all these facts really need recording, we may note that transcription itself is not an objective task but can be considered an editorial task dependent on the editor's erudition.[45] Second, Van Zundert's invitation to make the edition a richer experience with mixed media begs the question: why? Certainly, in scholarly editing of premodern texts, illustrations, sound, and video are hardly ever needed. It makes no sense to include them for the sake of inclusion. Third, there is a strange sense of demanding ever-more realistic representations of the original documents, ending in demand for facsimile photos. The point of an edition is exactly to move beyond the manuscripts in order to facilitate easier access. The advocates of digital editions laud the democratizing nature of the digital world, claiming that now manuscripts are not exclusively for the scholarly elite.[46] But restricting access, I would contend, was never the point. Whereas certain documents

42 Robinson, P. "The Digital Revolution in Scholarly Editing." pp. 181–207 in *Ars Edendi Lecture Series, Vol. IV,* edited by B. Crostini, G. Iversen, and B.M. Jensen. Stockholm: Stockholm University Press, 2016, p. 198.

43 Robinson, P. "Current Issues in Making Digital Editions of Medieval Texts -Or, Do Electronic Scholarly Editions Have a Future?" *Digital Medievalist* 1 (2005).

44 Robinson, "The Digital Revolution …", p. 193.

45 Pierazzo, p. 83.

46 Burdick et al, pp. 80, p. 87; Robinson, p. 198.

from the early modern period can be understood and found interesting by the general public, this is simply not the case for the vast majority of manuscripts, which require special training to be deciphered and a long study trajectory to be found meaningfully interesting. The task of editing such texts is exactly to make that training unnecessary and that trajectory considerably shorter. An example Robinson likes is Prue Shaw's edition of Dante's *Commedia*, which he lauds for not actually providing an edited text. "This absence strikes me as the single most remarkable element of the edition," he says.[47] While there certainly is merit in what he argues for, the unwillingness to let the editor make a scholarly argument in favor of a particular reading means we head in the direction of a mere repository. Apollon et al., in their *Digital Critical Editions*, argue specifically against Robinson. They make the obvious but highly pertinent remark that throwing together a bunch of transcriptions with markup and images "does not facilitate reading,"[48] and explain that "quantity rapidly becomes a problem to manage at both the level of display as well as the level of codification." This, of course, goes against the very nature of editing. Editing is about deciding which particular bits to pick out of the rich tapestry of the actual evidence to make the original text maximally understandable with minimum distractions. Simply passing on all of it is not helpful. Pierazzo makes a good and precise point when she says that "the easier it becomes to publish on the Web, the larger the quantity of textual materials available, the more important becomes the guidance of the editor. Quality still matters."[49]

Quality does indeed matter, and it matters the most if you want your edition to be taken seriously. What Robinson and Van Zundert argue for, however, puts quality in doubt in two ways. For one, by allowing anybody to contribute to the transcription (crowdsourcing), quality is hard to ascertain and even harder to be equal across the document. Further, by allowing changes after the first release, quality can change over time. Even though these aspects make a digital critical edition typically digital and do not jeopardize its nature of being an edition, they are exactly those aspects that make them less critical. Pierazzo points out that if an edition can change over time, reviewing becomes meaningless.[50] It seems that scholars know this intuitively, as there seems to

47 Robinson, "The Digital Revolution …", p. 196.
48 Apollon et al, p. 21.
49 Pierazzo, p. 8. Note however that Pierazzo nuances (perhaps contradicts) herself elsewhere when she makes the point that "digital editions based on text encoding allow the luxury of not choosing." cf. Pierazzo, E. "Textual Scholarship and Text Encoding." pp. 307–321 in *A New Companion to Digital Humanities*, edited by S. Schreibman, R. Siemens, and J. Unsworth. Oxford: Wiley-Blackwell, 2016, p. 313.
50 Pierazzo, p. 143.

be hesitation in citing digital editions. Indicative is Melissa Terras' answer to her own question regarding how a digital edition project, *Transcribe Bentham*, can benefit her: not by contributing to it but by using it in conventual paper publications.[51] Robinson himself admits that electronic publications are not seen as 'real.'[52] Burdick et al. think that DH should come up with new "evaluative metrics for legitimizing and credentialing this kind of scholarship,"[53] but this will be exceedingly hard given that digital editions are often very different from each other. This additional aspect makes digital editions odd indeed. For how can one evaluate something if it is unique and incomparable? Print publications have much more in common with each other, and this makes for easier comparisons as the format itself is reliable. It might just become a factor at the most crucial moments of your career when your output is scrutinized and compared to others because you are up for a job or tenure.[54]

Overseeing the pros and cons of digital critical editions, it seems as though we have a 'triple constraint' going on, in which we have 'digital,' 'critical,' and 'edition' and can only pick two. It would be a good thing if we saw a tandem model emerge, in which a scholar or a team of scholars produce both a (paper) critical edition and a digital (less critical) edition, especially in cases where aspects such as multiple layers of text reuse and internal referencing play a large role, which can be made much clearer through a digital edition. Such a model could foster a conversation to figure out exactly how a digital critical edition can fit into our discourse. Experimentation is crucial here. With the things discussed in this chapter, together with the next chapter, you should be able to pull together a web-based edition and experiment for yourself.

51 Terras, M. "Present, Not Voting: Digital Humanities in the Panopticon." pp. 172–190 in *Understanding Digital Humanities*, edited by D.M. Berry. New York: Palgrave Macmillan, 2012, pp. 179–180.
52 Robinson, "Current issues …"
53 Burdick et al, p. 84.
54 Cf. Prescott, A. "Consumers, Creators or Commentators? Problems of Audience and Mission in the Digital Humanities." pp. 61–75 in *Arts & Humanities in Higher Education* 11, no. 1–2 (2011), p. 62.