The Open Digital Archaeology Textbook Environment

Shawn Graham, Neha Gupta, Michael Carter, & Beth Compton

2017-02-22

Contents

notice **5**

About the Authors **7**

Getting Started **9**
How to use this text . . . 9
How to contribute changes, or make your own version . . . 10
How to access and use the computational environment . . . 10
Colophon . . . 10

Welcome! **13**

1 Going Digital **15**
1.1 So what is Digital Archaeology? . . . 15
1.2 Project management basics . . . 18
1.3 Github & Version Control . . . 20
1.4 Open Notebook Research & Scholarly Communication . . . 27
1.5 Failing Productively . . . 29
1.6 Introduction to Digital Libraries, Archives & Repositories . . . 30
1.7 Command Line Methods for Working with APIs . . . 30
1.8 The Ethics of Big Data in Archaeology . . . 33

2 Making Data Useful **35**
2.1 Designing Data Collection . . . 35
2.2 Cleaning Data with Open Refine . . . 35
2.3 Linked Open Data and Data Publishing . . . 35

3 Finding and Communicating the Compelling Story **37**
3.1 Statistical Computing with R and Python Notebooks; Reproducible code . . . 37
3.2 D3, Processing, and Data Driven Documents . . . 37
3.3 Storytelling and the Archaeological CMS: Omeka, Kora . . . 37
3.4 Web Mapping with Leaflet . . . 38
3.5 Place-based Interpretation with Locative Augmented Reality . . . 38
3.6 Archaeogaming and Virtual Archaeology . . . 38
3.7 Social media as Public Engagement & Scholarly Communication in Archaeology . . . 38

4 Eliding the Digital and the Physical **39**
4.1 3D Photogrammetry & Structure from Motion . . . 39
4.2 3D Printing, the Internet of Things and "Maker" Archaeology . . . 39
4.3 Artificial Intelligence in Digital Archaeology . . . 39

5 Digital Archaeology's Place in the World **41**
5.1 Marketing Digital Archaeology . . . 41

5.2 Sustainability & Power in Digital Archaeology 41

6 On the Horizons: Where Digital Archaeology Might Go Next 43

References 45

notice

This volume goes hand-in-glove with a computational environnent built on the DHBox.

THIS IS A DRAFT VERSION

Figure 1: A word cloud image of the original ODATE proposal, arranged to mimic a photograph of the temple of Athena Pronaos at Delphi

The online version of this book is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

About the Authors

Shawn Graham

Shawn Graham trained in Roman archaeology but has become over the years a digital archaeologist and digital humanist. He is currently an associate professor in the Department of History at Carleton University in Ottawa Canada. He keeps an open lab notebook of his research and experiments in digital history and archaeology at his research blog, electricarchaeology. He can be found on Twitter at electricarchaeo. His teaching explores methods and digital approaches at all levels, including seminars in the collaborative MA Digital Humanities program as well as in the MA Public History program. His research interests include augmented reality and virtual reality in the service of landscape archaeology, video games about the past, and agent based modeling.

Neha Gupta

blah

Michael Carter

blah

Beth Compton

blah

Editorial Board

Katharine Cook, University of Victoria

Ethan Watrall, Michigan State University

Daniel Pett, The British Museum

Eric Kansa, Open Context & The Alexandria Archive Institute

Kathleen Fitzpatrick, Modern Language Association

Getting Started

We wrote this text with a particular kind of student in mind. We imagined this student as having already taken a first year introductory course to archaeology, of the kind that surveys the field, its history, and its principle methods and theoretical positions. Very few courses of that kind include any depth in digital methods and theory, which is understandable when we look at the incredible variety of archaeological work, skills, and interests! Digital work is every bit as diverse as other kinds of archaeology, but it also presents its own particular challenges. One of these is the anxiety that comes when one first approaches the computer for anything more complex than word processing or a bit of social media. 'What happens if I break it?'; 'I'm not techy!'; 'If I wanted to do computers, I wouldn't have gone into this!' are all actual student concerns that we have heard in our various classrooms.

It'll be ok.

We take a pedagogical perspective that focuses on the learning that happens when we make things, when we experiment or otherwise play around with materials and computing, and especially, when/if things break. It's a perspective that finds value in 'failing gloriously', in trying to push ourselves beyond our comfort level. The only thing that you will need therefore to be successful in learning some of the basic issues around digital archaeology is a willingness to consider why things didn't work the way they ought to have, and a web browser. We built this textbook with its very own digital archaeology computer built right in! There's nothing you can break on your own machine, nor does your machine have to be very powerful. If you want, you can even download and install ODATE on a desktop computer to use as your very own digital archaeology laboratory.

How to use this text

Each section in this book is broken down into an overview or discussion of the main concepts, and then followed up with skill-building exercises. The computational environment, when you log into it, is set to expire after three hours - so you really ought to work through the sections on Github & Version Control and Open Notebook Research & Scholarly Communication so you'll be able to get your work and data out of ODATE and onto space that you control. The best way to use this book is to make sure you have at least one hour blocked out to read through a section, and then two hours to go through the section again if you're working on the exercises.

Do you notice that stripe down the side of the screen at the right? That's a tool-bar for annotating the text, using a tool called `Hypothes.is`. If you highlight any text (go ahead, highlight that phrase right now by right-clicking and dragging your mouse!) a little pop-up will ask you if you want to annotate or highlight the text. If you choose annotate, a writing pane will open on the right. Using the Hypothesis tool requires a reader to create a login and account with Hypothesis, which is managed by the Hypothesis site, not us.

By default, such annotations are made public. Private annotations can only be viewed by the particular individual who made them. All annotations (both public and private) have their own unique URL and can be collated in various ways using the Hypothesis API (here's an example). Please tag your annotation with `odate` to allow easy curating of the public annotations.

Please note that any public annotations can be read by any other reader. These can also be responded to, as well - which might make a great classroom activity! A class can create group annotations which are only visible to participants in that group. Annotation is a tool for research; personal reaction to anything we've written in ODATE should be done via the reader's blog while leaving an annotation on ODATE linking to the blog piece.

Over time, the parts of the text that are heavily annotated will look as if someone has gone over them with a yellow highlighter. You can use this to help guide your reading - perhaps that's a part where many people had problems, or perhaps it's a part that sparked a really interesting discussion! Group annotation like this promotes 'active reading', which means that you're more likely to retain the discussion.

Finally, if you'd rather not read this as a web page, you can grab a pdf copy by pressing the download button above (the downwards-facing arrow icon) and printing out just the bits you want or need. If you'd rather read this text via an e-reader or iBooks or similar, the download link will also give you an ePub version. Individuals who use a screenreader or other assistive device might prefer to work with the pdf or epub versions. Please do let us know if there is any way we can make this text more accessible for users with particular needs. Since this text is fundamentally a series of plain-text files that we then manipulate to create these different outputs, it should be straightforward for us to adapt accordingly!

How to contribute changes, or make your own version

Perhaps your professor has assigned a portion of this text to your class, with the instruction to improve it. Do you see the edit button at the top of the screen (it looks like a little square with a pencil)? If you click on that, and you have an account on Github (and you're signed in), you will grab a copy of this entire text (but not the computational part; the source code for that is here instead) that you can then edit. If you want, you can also make a `pull-request` to us, asking us to fold your changes into our textbook. We welcome these suggestions! Since this book has a creative-commons license, you are welcome to expand and build upon this as you wish, but do cite and link back to the original version.

How to access and use the computational environment

link to site, instructions, also repo, also dhbox-on-a-stick

Colophon

This text was created using the Bookdown package for R Markdown. R Markdown is a variant of the simple Markdown format created by John Gruber. That is to say, at its core this text is a series of simple text-files marked up with simple markers of syntax like `#` marks to indicate headings and so on. R Markdown allows us to embed code snippets within the text that an interpreter, like R Studio, knows how to run, such that the results of the calculations become embedded in the surrounding discussion! This is a key part of making research more open and more reproducible, and which you'll learn more about in chapter one.

The sequence of steps to produce a Bookdown-powered site looks like this:

1. create a new project in RStudio (we typically create a new project in a brand new folder)
2. run the following script to install Bookdown:

```
install.packages("devtools")
devtools::install_github("rstudio/bookdown")
```

3. create a new textfile with `metadata` that describe how the book will be built. The metadata is in a format called `YAML` ('yet another markup language') that uses keys and values that get passed into other parts of Bookdown:

```
title: "The archaeology of spoons"
author: "Graham, Gupta, Carter, & Compton"
date: "July 1 2017"
description: "A book about cocleararchaeology."
github-repo: "my-github-account/my-project"
cover-image: "images/cover.png"
url: 'https\://my-domain-ex/my-project/'
bibliography: myproject.bib
biblio-style: apa-like
link-citations: yes
```

This is the only thing you need to have in this file, which is saved in the project folder as `index.Rmd`.

4. Write! We write the content of this book as text files, saving the parts in order. Each file should be numbered `01-introduction.Rmd`, `02-a-history-of-spoons.Rmd`, `03-the-spoons-of-site-x.Rmd` and so on.

5. Build the book. With Bookdown installed, there will be a 'Build Book' button in the R Studio build pane. This will generate the static html files for the book, the pdf, and the epub. All of these will be found in a new folder in your project, `_book`. There are many more customizations that can be done, but that is sufficient to get one started.

The computational environment

The computational environment we are using is a fork of the DhBox project from CUNY, led by Stephen Zweibel, Patrick Smyth Developer, Jojo Karlin, and Matthew K. Gold. DHBox requires Ubuntu >= 14.04 and Python 2.7x. We run it at Carleton courtesy of the School of Computer Science, Andrew Pullin, Peter Choynowski, and Doug Howe, via OpenStack and Docker. A version of DHBox that can be run from a USB stick can be found here and is worth exploring!

Welcome!

Digital archaeology as a field rests upon the creative use of primarily open-source and/or open-access materials to archive, reuse, visualize, analyze and communicate archaeological data. This reliance on open-source and open-access is a political stance that emerges in opposition to archaeology's past complicity in colonial enterprises and scholarship; digital archaeology resists the digital neo-colonialism of Google, Facebook, and similar tech giants that typically promote disciplinary silos and closed data repositories. Specifically, digital archaeology encourages innovative, reflective, and critical use of open access data and the development of digital tools that facilitate linkages and analysis across varied digital sources.

To that end, this document you are reading is integrated with a cloud-based digital exploratory laboratory of multiple cloud-computing tools with teaching materials that instructors will be able to use 'out-of-the-box' with a single click, or to remix as circumstances dictate. Part of our inspiration comes from the 'DHBox' project from CUNY (City University of New York, (link), a project that is creating a 'digital humanities laboratory' in the cloud. While the tools of the digital humanities are congruent with those of digital archaeology, they are typically configured to work with texts rather than material culture in which archaeologists specialise. The second inspiration is the open-access guide 'The Programming Historian', which is a series of how-tos and tutorials (link) pitched at historians confronting digital sources for the first time. A key challenge scholars face in carrying out novel digital analysis is how to install or configure software; each 'Programming Historian' tutorial therefore explains in length and in detail how to configure software. The present e-textbook merges the best of both approaches to create a singular experience for instructors and students: a one-click digital laboratory approach, where installation of materials is not an issue, and with carefully designed tutorials and lessons on theory and practice in digital archaeology.

Chapter 1

Going Digital

> Digital archaeology should exist to assist us in the performance of archaeology as a whole. It should not be a secret knowledge, nor a distinct school of thought, but rather simply seen as archaeology done well, using all of the tools available to and in better recovering, understanding and presenting the past. In the end, there is no such thing as digital archaeology. What exists, or at least what should exist, are intelligent and practical ways of applying the use of computers to archaeology that better enable us to pursue both our theoretical questions and our methodological applications. (Evans and Daly 2006)

While we agree with the first part of the sentiment, the second part is rather up for debate. Digital tools exist in a meshwork of legal and cultural obligations, and moreso than any other tool humans have yet come up with, have the capability to exert their own agency upon the user. Digital tools and their use are not theory-free or without theoretical implications. There is no such thing as neutral, when digital tools are employed.

In this section, we suggest that digital archaeology is akin to work at the intersection of art and public archaeology and digital humanities. We then provide you the necessary basics for setting up your own digital archaeological practice.

1.1 So what is Digital Archaeology?

If you are holding this book in your hands, via a device or on paper, or looking at it on your desktop, you might wonder why we feel it necessary to even ask the question. It is important at the outset to make the argument that digital archaeology is not about 'mere' tool use. Andrew Goldstone in *Debates in the Digital Humanities* discusses this tension (Goldstone 2018). He has found (and Lincoln Mullen concurs with regard to his own teaching,(Mullen 2017)) that our current optimism about teaching technical facility is misplaced. Tools first, context second doesn't work. Alternatively, theory first doesn't seem to work either. And finally, for anything to work at all, datasets have to be curated and carefully pruned for their pedagogical value. We can't simply turn students loose on a dataset (or worse, ask them to build their own) and expect 'learning' to happen.

Our approach in this volume is to resolve that seeming paradox by providing not just the tools, and not just the data, but also the computer itself. Archaeologically, this puts our volume in dialog with the work of scholars such as Ben Marwick, who makes available with his research the code, the dependencies, and sometimes, an entire virtual machine, to enable other scholars to replicate, reuse, or dispute his conclusions. We want you to reuse our code, to study it, and to improve upon it. We want you to annotate our pages, and point out our errors. For us, digital archaeology is not the mere use of computational tools to answer archaeological questions. Rather, it is to enable the audience for archaeological thinking to enter into conversation with us, and to *do* archaeology for themselves.

Digital archaeology is necessarily a public archaeology. This is its principal difference with what has come before, for never forget, there has been at least a half-century of innovative use of computational power for archaeological knowledge building.

1.1.1 Is digital archaeology part of the digital humanities?

In recent years - certainly the last decade - an idea called 'the digital humanities' has been percolating around the academy. It is a successor idea to 'humanities computing', but it captures that same distinction between discussed above. Digital archaeology has developed alongside the digital humanities, sometimes intersecting with it (notably, there was a major archaeological session at the annualinternational Alliance of Digital Humanities Organizations (ADHO) DH conference in 2013). The various component organizations of the ADHO have been meeting in one form or another since the 1970s; so too the Computer Applications in Archaeology Conference has been publishing its proceedings since 1973. Archaeologists have been running simulations, doing spatial analysis, clustering, imaging, geophysicing, 3d modeling, neutron activation analyzing, x-tent modeling , etc, for what seems like ages. Happily, there is no one definition of 'dh' that everyone agrees on (see the various definitions collected at http://definingdh.org/; reload the page to get a new definition). For us, a defining *characteristic* of DH work is that public use we discussed above. But, another characteristic that we find useful to consider is the *purpose* to which computation is put in DH work.

Trevor Owens, a digital archivist, draws attention to the purpose behind one's use of computational power – generative discovery versus justification of an hypothesis (Owens, Trevor 2012). Discovery marks out the digital humanist whilst justification signals the humanist who uses computers. Discovery and justification are critically different concepts. For Owens, if we are using computational power to deform our texts, then we are trying to see things in a new light, to create new juxtapositions, to spark new insight. Stephen Ramsay talks about this too in Reading Machines (Ramsay 2011, 33), discussing the work of Samuels and McGann, (Samuels and McGann 1999): "Reading a poem backward is like viewing the face of a watch sideways – a way of unleashing the potentialities that altered perspectives may reveal". This kind of reading of data (especially, but not necessarily, through digital manipulation), does not happen very much at all in archaeology. If 'deformance' is a key sign of the digital humanities, then digital archaeologists are not digital humanists. Owen's point isn't to signal who's in or who's out, but rather to draw attention to the fact that:

> When we separate out the the context of discovery and exploration from the context of justification we end up clarifying the terms of our conversation. There is a huge difference between "here is an interesting way of thinking about this" and "This evidence supports this claim."

This is important in the wider conversation concerning how we evaluate digital scholarship. We've used computers in archaeology for decades to try to justify or otherwise connect our leaps of logic and faith, spanning the gap between our data and the stories we'd like to tell. We believe, on balance, that 'digital archaeology' sits along this spectrum between justification and discovery closer to the discovery end, that it sits within the digital humanities and should worry less about hypothesis testing, and concentrate more on discovery and generation, of 'interesting way[s] of thinking about this'.

Digital archaeology should be a prompt to make us 'think different'. Let's take a small example of how that might play out. It's also worth suggesting that 'play' as a strategy for doing digital work is a valid methodology (see Ramsay (2011)).

1.1.2 Archaeological Glitch Art

Bill Caraher is a leading thinker on the implications and practice of digital archaeology. In a post on archaeological glitch art (Caraher 2012) Caraher changed file extensions to fiddle about in the insides of images of archaeological maps. He then looked at them again as images:

> The idea ... is to combine computer code and human codes to transform our computer mediated image of archaeological reality in unpredictable ways. The process is remarkably similar to analyzing the site via the GIS where we take the "natural" landscape and transform it into a

> series of symbols, lines, and text. By manipulating the code that produces these images in both random and patterned ways, we manipulate the meaning of the image and the way in which these images communicate information to the viewer. We problematize the process and manifestation of mediating between the experienced landscape and its representation as archaeological data.

Similarly, Graham's work in representing archaeological data in sound (a literal auditory metaphor) translates movement over space (or through time) into a soundscape of tones (Graham 2017). This frees us from the tyranny of the screen and visual modes of knowing that often occlude more than they reveal (for instance, our Western-framed understanding of the top of the page or screen as 'north' means we privilege visual patterns in the vertical dimension over the horizontal (Montello et al. 2003)).

These playful approaches force us to rethink some of our norms of communication, our norms of what archaeology can concern itself with. It should be apparent that digital archaeology transcends mere 'digital skills' or 'tool use'; but it also suffers from being 'cool'.

1.1.3 The 'cool' factor

Alan Liu (Liu 2004) wondered what the role of the arts and humanities was in an age of knowledge work, of deliverables, of an historical event horizon that only goes back the last financial quarter. He examined the idea of 'knowledge work' and teased out how much of the driving force behind it is in pursuit of the 'cool'. Through a deft plumbing of the history of the early internet (and in particular, riffing on Netscape's 'what's cool?' page from 1996 and their inability to define it except to say that they'd know it when they saw it), Liu argues that cool is 'the aporia of information... cool is information designed to resist information... information fed back into its own signal to create a standing interference pattern, a paradox pattern' (Liu 2004, 179). The latest web design, the latest app, the latest R package for statistics, the latest acronym on Twitter where all the digital humanists play: cool, and dividing the world.

That is, Liu argued that 'cool' was amongst other things a politics of knowledge work, a practice and ethos. He wondered how we might 'challenge knowledge work to open a space, as yet culturally sterile (coopted, jejune, anarchistic, terroristic), for a more humane hack of contemporary knowledge?' (Liu 2004, 9). Liu goes on to discuss how the tensions of 'cool' in knowledge work (for us, read: digital archaeology) also intersects with an ethos of the unknown, that is, of knowledge workers who work nowhere else somehow manage to stand outside that system of knowledge production. (Is alt-ac 'alt' partially because it is the cool work?). This matters for us as archaeologists. There are many 'cool' things happening in digital archaeology that somehow do not penetrate into the mainstream (such as it is). The utilitarian dots-on-a-map were once cool, but are now pedestrian. The 'cool' things that could be, linger on the fringes. If they did not, they wouldn't be cool, one supposes. They resist.

To get that more humane hack that Liu seeks, Liu suggests that the historical depth that the humanities provides counters the shallowness of cool:

> The humanities thus have an explanation for the new arts of the information age, whose inheritance of a frantic sequence of artistic modernisms, postmodernisms, and post-postmodernists is otherwise only a displaced encounter with the raw process of historicity. Inversely, the arts offer the humanities serious ways of engaging – both practically and theoretically- with "cool". Together, the humanities and arts might be able to offer a persuasive argument for the humane arts in the age of knowledge work. (Liu 2004, 381).

In which case, the emergence of digital archaeologists and historians in the last decade might be the loci of the humane hacks – if we move into that space where we engage the arts. Indeed, the seminal anthropologist Tim Ingold makes this very argument with reference to his own arc as a scholar, 'From Science to Art and Back Again':

> Revisiting science and art: which is more ecological now? Why is art leading the way in promoting radical ecological awareness? The goals of today's science are modelling, prediction and control. Is that why we turn to art to rediscover the humility that science has lost?

We need to be making art. Digital archaeology naturally pushes in that direction.

1.1.4 Takeaways

- Digital archaeology is a public archaeology
- Digital archaeology is more often about deformance rather than justification
- In that deformative practice, it is in some ways extremely aligned with artistic ways of knowing
- Digital archaeology is part of the digital humanities, and in many ways, presaged current debates and trends in that field.

All of these aspects of digital archaeology exist along a continuum. In the remainder of this chapter, we give you a 'boot-camp' to get you to the point where you can begin to wonder about deformation and the public entanglement with your work.

1.2 Project management basics

A digital project, whether in archaeology or in other fields, iterates through the same basic steps. There is

```
1. finding data
2. fixing data
3. analyzing the data
4. communicating the story in the data
```

Eighty percent of your time on any digital project will be invested in cleaning up the data and documenting what you've done to it. But in truth, a digital project begins long before we ever look at a data set (or are given data to work with, if we're part of a larger project). How do we formulate a research question or our exploration more generally? How do we translate a gut feeling or intution or curiosity into something that is *operable*? REF Moretti on operationalizing things

Firstly, we have to develop some basic good habits.

```
1. separate what your write from _how_ you write it.
2. keep what you write under version control.
```

Have you ever fought with Word or another wordprocessor, trying to get things just right? Word processing is a mess. It conflates writing with typesetting and layout. Sometimes, you just want to get the words out. Othertimes, you want to make your writing as accessible as possible... but your intended recipient can't open your file, because they don't use the same wordprocessor. Or perhaps you wrote up some great notes that you'd love to have in a slideshow; but you can't, because copying and pasting preserves a whole lot of extra gunk that messes up your materials.

The answer is to separate your content from your tool. This can help keep your thinking clear, but it also has a more nuts-and-bolts practical dimension. *A computer will always be able to read a text file.* That is to say: you've futureproofed your material. Any researcher will have old physical discs or disc drives or obsolete computers lying around. It is not uncommon for a colleague to remark, 'I wrote this in Wordperfect and I can't open this any more'. Graham's MA thesis is trapped on a 3.5" disc drive that was compressed using a now-obsolete algorithm and it cannot be recovered. If, on the other hand, he had written the text as a .txt file, and saved the data as .csv tables, those materials would *continue* to be accessible.

(a .txt file is simply a text file; a .csv is a text file that uses commas to separate the text into columns. Similarly, a .md file is a text file that uses things like `#` to indicate headers, and `_` to show where italicized text starts and stops.)

As you work through this book, we encourage you to write your thoughts, observations, or results in simple text files. You can always open a text file in the latest version of Word; sometimes, you can't open old Word

files in the latest version of Word! We will discuss more about this below, where we will have you do some exercises in marking up a text file with 'markdown'.

The four steps we identified above are cyclical; at any one time you might be at a different stage of the process. Indeed, those four steps could easily be subsumed under what Simon Appleford and Jennifer Guiliano of devdh.org identify as the 'Best Practice Principles Of Designing Your First Project.' For Appleford and Guiliano, the outline of a project involves figuring out:

1. the question, problem, or provocation
2. sources (primary, secondary)
3. analytical activity
4. audience
5. product

Note that 4, audience, comes before 5, product. You must think of your reader/user!

Let us imagine that we were inspired by Allison Mickel's piece, 'Tracing Teams, Texts, and Topics: Applying Social Network Analysis to Understand Archaeological Knowledge Production at Çatalhöyük'.

We could frame a question: 'What role do social networks play in the development of knowledge production at my site?'

We could frame a problem: 'Mickel's exploration of social networks considered x, but not y.'

We could frame a provocation: 'Social Network Analysis promises to revolutionize our knowledge of the social contexts that underpin archaeological fieldwork, putting this power in the hands of everyone from the site director on down.'

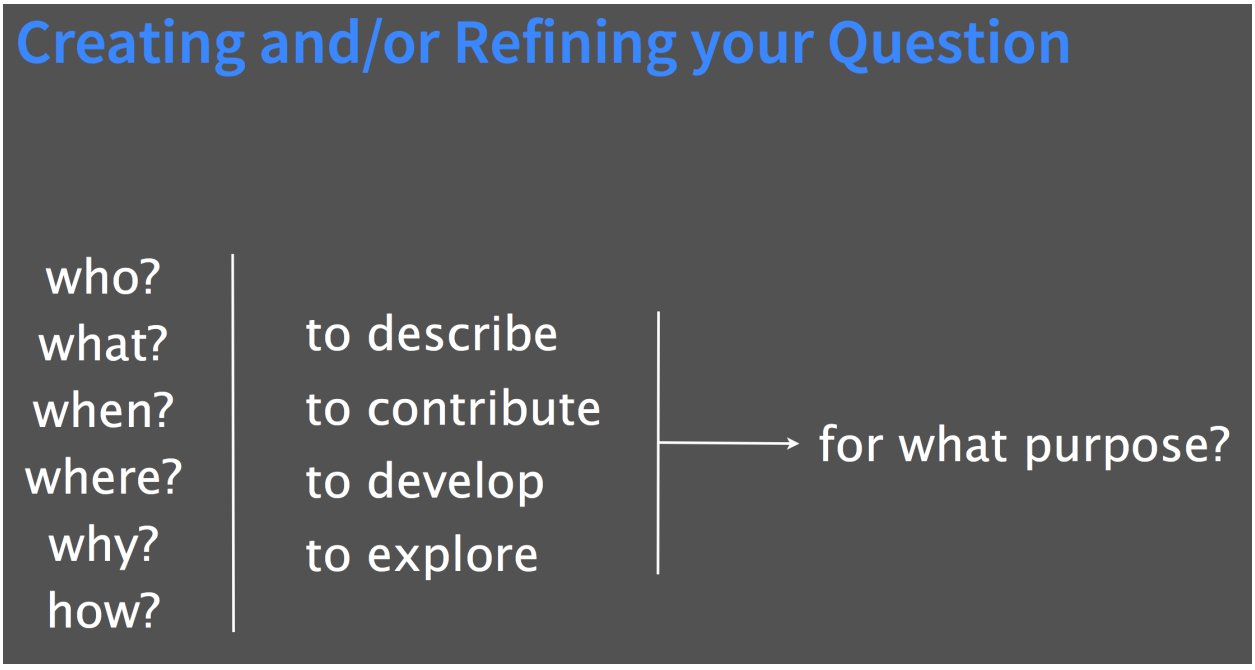

Figure 1.1: Creating and/or refining your research question, per DevDH.org

Following Appleford and Guiliano, we can refine our question, or our problem, or our provocation down to its essence in order to figure out the next parts of the the process. Knowing exactly what kind of question, problem, or provocation we're after, we then have a better sense of what to do when confronted with a mass of data (for instance, the excavation diaries from Kenen Tepe held in OpenContext.org, deposited by Parker and Cobb, 2012). Once the question is well-drawn out, questions 3 and 4 take care of themselves.

One other element that we might add is 'collaboration'. How do you plan to collaborate? While many digital archaeology projects are done by a single individual working in the quiet of their own space, most projects require many different skill sets, perspectives, and stakeholders. It is worth figuring out at the outset how you plan to work together. Will you use email? Will you use a private slack or messaging client? What about Kanban boards? (A Kanban board can be as simple as a whiteboard with three columns on it, marked 'to do', 'doing', and 'done'. Tasks are written on post-it notes and moved through the columns as necessary. A popular software implementation of a Kanban board is Trello.) We would also recommend that you write down the ideal division of labour and areas of responsibility for each of the participants, *along with a mechanism for resolving disputes.*

Finally, how much time would you have to work on your digital archaeology project? All of us have multiple demands on our time. Let's be realistic about how much time you have available. How many hours, total, do you spend in class, at work, asleep, and socializing? Add that up for a week, then multiply by the number of weeks in your term. There are 384 hours in a 16 week term. Subtract to find out how many 'spare' hours you can devote to homework, this project, or a hobbby.

Divide that by the number of weeks your course runs. That's how many hours per week you can spend on all your non in-class course work. Then, divide those hours by the number of courses you have.

Assuming that this textbook is being used for one of those courses, that's how much time you would have to spend on a digital archaeology project. It's not an awful lot, which means that the more energy you put into planning, the more effective your labour is going to be.

1.2.1 Take-aways

- Separate your *content* from your *presentation*
- future-proof your materials through the use of simple text formats wherever possible
- be explicit about how collaboration will be managed
- be explicit about how your research goals intersect with your audience
- be brutally honest about your time and guard it jealously

1.2.2 exercises

1. (to be refined) Open a new textfile in the Archaebox. Call it 'initial-project-idea.md'. Using `#` to indicate headings, sketch out a question, problem, or provocation of your own that occurs to you as you browse the Kenen Tepe materials housed at OpenContext.org. Save that file.
2. create a project management plan. (to be fleshed out in more detail)

1.3 Github & Version Control

It's a familiar situation - you've been working on a paper. It's where you want it to be, and you're certain you're done. You save it as 'final.doc'. Then, you ask your friend to take a look at it. She spots several typos and that you flubbed an entire paragraph. You open it up, make the changes, and save as 'final-w-changes.doc'. Later that day it occurs to you that you don't like those changes, and you go back to the original 'final.doc', make some changes, and just overwrite the previous version. Soon, you have a folder like:

```
|-project
    |-'finalfinal.doc'
    |-'final-w-changes.doc'
    |-'final-w-changes2.doc'
    |-'isthisone-changes.doc'
    |-'this.doc'
```

Things can get messy quite quickly. Imagine that you also have several spreadsheets in there as well, images, snippets of code... we don't want this. What we want is a way of managing the evolution of your files. We do this with a program called Git. Git is not a user-friendly piece of software, and it takes some work to get your head around. Git is also very powerful, but fortunately, the basic uses to which most of us put it to are more or less straightforward. There are many other programs that make use of Git for version control; these programs weld a graphical user interface on top of the main Git program. It is better however to become familiar with the basic uses of git from the command line *first* before learning the idiosyncracies of these helper programs. The exercises in this section will take you through the basics of using Git from the command line.

1.3.1 The core functions of Git

Figure 1.2: Alice Bartlett's summary of what Git does

At its heart, Git is a way of taking 'snapshots' of the current state of a folder, and saving those snapshots in sequence. (For an excellent brief presentation on Git, see Alice Bartlett's presentation here; Bartlett is a senior developer for the Financial Times). In Git's lingo, a folder on your computer is known as a `repository`. This sequence of snapshots in total lets you see how your project unfolded over time. Each time you wish to take a snapshot, you make a `commit`. A commit is a Git command to take a snapshot of the entire repository. Thus, your folder we discussed above, with its proliferation of documents becomes:

```
|-project
    |-'final.doc'
```

BUT its commit history could be visualized like this:

Figure 1.3: A visualization of the history of commits

Each one of those circles represents a point in time when you the writer made a commit; Git compared the state of the file to the earlier state, and saved a snapshot of the `differences`. What is particularly useful

about making a commit is that Git requires two more pieces of information about the git: who is making it, and when. The final useful bit about a commit is that you can save a detailed message about *why* the commit is being made. In our hypothetical situation, your first commit message might look like this:

```
Fixed conclusion

Julie pointed out that I had missed
the critical bit in the assignment
regarding stratigraphy. This was
added in the concluding section.
```

This information is stored in the history of the commits. In this way, you can see exactly how the project evolved and why. Each one of these commits has what is called a `hash`. This is a unique fingerprint that you can use to 'time travel' (in Bartlett's felicitous phrasing). If you want to see what your project looked like a few months ago, you `checkout` that commit. This has the effect of 'rewinding' the project. Once you've checked out a commit, don't be alarmed when you look at the folder: your folder (your repository) looks like how it once did all those weeks ago! Any files written after that commit seem as if they've disappeared. Don't worry: they still exist!

What would happen if you wanted to experiment or take your project in a new direction from that point forward? Git lets you do this. What you will do is create a new `branch` of your project from that point. You can think of a branch as like the branch of a tree, or perhaps better, a branch of a river that eventually merges back to the source. (Another way of thinking about branches is that it is a label that sticks with these particular commits.) It is generally considered 'best practice' to leave your `master` branch alone, in the sense that it represents the best version of your project. When you want to experiment or do something new, you create a `branch` and work there. If the work on the branch ultimately proves fruitless, you can discard it. *But*, if you decide that you like how it's going, you can `merge` that branch back into your master. A merge is a commit that folds all of the commits from the branch with the commits from the master.

Git is also a powerful tool for backing up your work. You can work quite happily with Git on your own machine, but when you store those files and the history of commits somewhere remote, you open up the possibility of collaboration *and* a safe place where your materials can be recalled if -perish the thought- something happened to your computer. In Git-speak, the remote location is, well, the `remote`. There are many different places on the web that can function as a remote for Git repositories. You can even set one up on your own server, if you want. One of the most popular (and the one that we use for ODATE) is Github. There are many useful repositories shared via Github of interest to archaeologists - OpenContext for instance shares a lot of material that way. To get material *out* of Github and onto your own computer, you `clone` it. If that hypothetical paper you were writing was part of a group project, your partners could clone it from your Github space, and work on it as well!

You and Anna are working together on the project. You have made a new project repository in your Github space, and you have cloned it to your computer. Anna has cloned it to hers. Let's assume that you have a very productive weekend and you make some real headway on the project. You `commit` your changes, and then `push` them from your computer to the Github version of your repository. That repository is now one commit *ahead* of Anna's version. Anna `pulls` those changes from Github to her own version of the repository, which now looks *exactly* like your version. What happens if you make changes to the exact same part of the exact same file? This is called a `conflict`. Git will make a version of the file that contains text clearly marking off the part of the file where the conflict occurs, with the conflicting information marked out as well. The way to `resolve` the conflict is to open the file (typically with a text editor) and to delete the added Git text, making a decision on which information is the correct information.

1.3.2 Key Terms

- repository: a single folder that holds all of the files and subfolders of your project
- commit: this means, 'take a snapshot of the current state of my repostiory'

- publish: take my folder on my computer, and copy it and its contents to the web as a repository at github.com/myusername/repositoryname
- sync: update the web repository with the latest commit from my local folder
- branch: make a copy of my repository with a 'working name'
- merge: fold the changes I have made on a branch into another branch
- fork: to make a copy of someone else's repo
- clone: to copy an online repo onto your own computer
- pull request: to ask the original maker of a repo to 'pull' your changes into their master, original, repository
- push: to move your changes from your computer to the online repo
- conflict: when two commits describe different changes to the same part of a file

1.3.3 Take-aways

- Git keeps track of all of the differences in your files, when you take a 'snapshot' of the state of your folder (repository) with the `commit` command
- Git allows you to roll back changes
- Git allows you to experiment by making changes that can be deleted or incorporated as desired
- Git allows you to manage collaboration safely
- Git allows you to distribute your materials

1.3.4 Further Reading

We alluded above to the presence of 'helper' programs that are designed to make it easier to use Git to its full potential. An excellent introduction to Github's desktop GUI is at this Programming Historian lesson on Github. A follow-up lesson explains the way Github itself can be used to host entire websites! You may explore it here. In the section of this chapter on open notebooks, we will also use Git and Github to create a simple open notebook for your research projects.

You might also wish to dip into the archived live stream; link here from the first day of the NEH funded Institute on Digital Archaeology Method and Practice (2015) where Prof. Ethan Watrall discusses project management fundamentals and, towards the last part of the stream, introduces Git.

1.3.5 Exercises

1. How do you turn a folder into a repository? With the `git init` command. At the command line (remember, the `$` just shows you the prompt; you don't have to type it!):

a. make a new director: `$ mkdir first-repo`
b. type `$ ls` (list) to see that the director exists. Then change directory into it: `cd first-repo`. (remember: if you're ever not sure what directory you're in, type `$ pwd`, or print working directory).
c. make a new file called `readme.md`. You do this by calling the text editor: `nano readme.md`. Type an explanation of what this exercise is about. The `.md` signals that you're writing a text file that uses the markdown format of signalling things like headings, lists, tables, etc. (A guide to markdown syntax is here). Hit ctrl+x to exit, then y to save, leave the file name as it is.
d. type `$ ls` again to check that the file is there.
e. type `$ git init` to tell the Git program that this folder is to be tracked as a repository. If all goes correctly, you should see a variation on this message: `Initialized empty Git repository in /home/demonstration/first-repo/.git/`. But type `$ ls` again. What do you (not) see?

The changes in your repo will now be stored in that *hidden* directory, `.git`. Most of the time, you will never have reason to search that folder out. But know that the config file that describes your repo is in that folder. There might come a time in the future where you want to alter some of the default behaviour of the git

program. You do that by opening the config file (which you can read with a text editor). Google 'show hidden files and folders' for your operating system when that time comes.

2. Open your readme.md file again with the nano text editor, from the command line. Add some more information to it, then save and exit the text editor.

a. type `$ git status`
b. Git will respond with a couple of pieces of information. It will tell you which `branch` you are on. It will list any untracked files present or new changes that are unstaged. We now will `stage` those changes to be added to our commit history by typing `$ git add -A`. (the bit that says `-A` adds any new, modified, or deleted files to your commit when you make it. There are other options or flags where you add *only* the new and modified files, *or* only the modified and deleted files.)
c. Let's check our git status again: type `$ git status`
d. You should see something like this:

```
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   readme.md```
```

e. Let's take a snapshot: type `$ git commit -m "My first commit"`. What happened? Remember, Git keeps track not only of the changes, but *who* is making them. If this is your first time working with Git in the Archaebox, Git will ask you for your name and email. Helpfully, the Git error message tells you exactly what to do: type `$ git config --global user.email "you\@example.com"` and then type `$ git config --global user.name "Your Name"`. Now try making your first commit.
f. The command above represents a bit of a shortcut for making commit messages by using the `-m` flag to associate the text in the quotation marks with the commit. Open up your readme.md file again, and add some more text to it. Save and exit the text editor. Add the new changes to the snapshot that we will take. Then, type `$ git commit`. Git automatically opens up the text editor so you can type a longer, more substantive commit message. In this message (unlike in markdown) the `#` indicates a line to be ignored. You'll see that there is already some default text in there telling you what to do. Type a message indicating the nature of the changes you have made. Then save and exit the text editor. DO NOT change the filename!

Congratulations, you are now able to track your changes, and keep your materials under version control!

3. Go ahead and make some more changes to your repository. Add some new files. Commit your changes after each new file is created. Now we're going to view the history of your commits. Type `$ git log`. What do you notice about this list of changes? Look at the time stamps. You'll see that the entries are listed in reverse chronological order. Each entry has its own 'hash' or unique ID, the person who made the commit and time are listed, as well as the commit message eg:

```
commit 253506bc23070753c123accbe7c495af0e8b5a43
Author: Shawn Graham <shawn.graham@carleton.ca>
Date:   Tue Feb 14 18:42:31 2017 +0000

Fixed the headings that were broken in the about section of readme.md
```

a. We're going to go back in time and create a new branch. Here's how the command will look: `$ git checkout -b branchname <commit>` where `branch` is the name you want the branch to be called, and `<commit>` is that unique ID. Make a new branch from your second last commit (don't use < or >).
b. We typed `git checkout -b experiment 253506bc23070753c123accbe7c495af0e8b5a43`. The response: `Switched to a new branch 'experiment'` Check git status and then list the contents of your repository. What do you see? You should notice that some of the files you had created before seem to have disappeared - congratulations, you've time travelled! Those files are not missing; but they *are* on a different branch (the master branch) and you can't harm them now. Add a number of new files,

making commits after each one. Check your git status, and check your git log as you go to make sure you're getting everything. Make sure there are no unstaged changes - everything's been committed.

4. Now let's assume that your `experiment` branch was successful - everything you did there you were happy with and you want to integrate all of those changes back into your `master` branch. We're going to merge things. To merge, we have to go back to the master branch: `$ git checkout master`. (Good practice is to keep separate branches for all major experiments or directions you go. In case you lose track of the names of the branches you've created, this command: `git branch -va` will list them for you.)

a. Now, we merge with `$ git merge experiment`. Remember, a merge is a special kind of commit that rolls all previous commits from both branches into one - Git will open your text editor and prompt you to add a message (it will have a default message already there if you want it). Save and exit and ta da! Your changes have been merged together.

5. One of the most powerful aspects of using Git is the possibility of using it to manage collaborations. To do this, we have to make a copy of your repository available to others as a `remote`. There are a variety of places on the web where this can be done; one of the most popular at the moment is Github. Github allows a user to have an unlimited number of `public` repositories. Public repositories can be viewed and copied by anyone. `Private` repositories require a paid account, and access is controlled. If you are working on sensitive materials that can only be shared amongst the collaborators on a project, you should invest in an upgraded account (note that you can also control which files get included in commit; see this help file. In essence, you simply list the file names you do not want committed; here's an example). Let's assume that your materials are not sensitive.

a. Go to Github, register for an account.

b. On the upper right part of the screen there is a large + sign. Click on that, and select `new public repository`

c. On the following screen, give your repo a name.

d. DO NOT 'initialize this repo with a readme.md'. Leave `add .gitignore` and `add license` set to NONE.

e. Clic the green 'Create Repository' button.

f. You now have a space into which you will publish the repository on your machine. At the command line, we now need to tell Git the location of this space. We do that with the following command, where you will change `your-username` and `your-new-repo` appropriately:

```
$ git remote add origin https://github.com/YOUR-USERNAME/YOUR-NEW-REPO.git
```

g. Now we push your local copy of the repository onto the web, to the Github version of your repo:

```
git push -u origin master
```

NB If you wanted to push a `branch` to your repository on the web instead, do you see how you would do that? If your branch was called `experiment`, the command would look like this:

```
$ git push origin experiment
```

h. The changes can sometimes take a few minutes to show up on the website. Now, the next time you make changes to this repository, you can push them to your Github account - which is the 'origin' in the command above. Add a new text file. Commit the changes. Push the changes to your account.

6. Imagine you are collaborating with one of your classmates. Your classmate is in charge of the project, and is keeping track of the 'official' folder of materials (eg, the repo). You wish to make some changes to the files in that repository. You can manage that collaboration via Github by making a copy, what Github calls a `fork`.

a. Make sure you're logged into your Github account on the Github website. We're going to fork an example repository right now by going to https://github.com/octocat/Spoon-Knife. Click the 'fork' button at top-right. Github now makes a copy of the repository in your own Github account!

b. To make a copy of that repository on your own machine, you will now clone it with the `git clone` command. (Remember: a 'fork' copies someone's Github repo into a repo in your OWN Github account; a 'clone' makes a copy on your own MACHINE). Type:

```
$ cd..
$ pwd
```

We do that to make sure you're not *inside* any other repo you've made! Make sure you're not inside the repository we used in exercises 1 to 5, then proceed:

```
$ git clone https://github.com/YOUR-USERNAME/Spoon-Knife
$ ls
```

You now have a folder called 'Spoon-Knife' on your machine! Any changes you make inside that folder can be tracked with commits. You can also `git push -u origin master` when you're inside it, and the changes will show up on your OWN copy (your fork) on Github.com. c. Make a fork of, and then clone, one of your classmates' repositories. Create a new branch. Add a new file to the repository on your machine, and then push it to your fork on Github. Remember, your new file will appear on the new branch you created, NOT the master branch.

7. Now, you let your collaborator know that you've made a change that you want her to `merge` into the original repository. You do this by issuing a `pull request`. But first, we have to tell Git to keep an eye on that original repository, which we will call `upstream`. You do this by adding that repository's location like so:

a. type (but change the address appropriately):

```
$ git remote add upstream THE-FULL-URL-TO-THEIR-REPO-ENDING-WITH-.git
```

b. You can keep your version of the remote up-to-date by fetching any new changes your classmate has done:

```
$ git fetch upstream
```

c. Now let's make a `pull` request (you might want to bookmark this help document). Go to your copy of your classmate's repository at your Github account. Make sure you've selected the correct branch you pushed your changes to, by selecting it from the Branches menu drop down list.

d. Click the 'new pull request' button.

e. The new page that appears can be confusing, but it is trying to double check with you which changes you want to make, and where. *'Base Branch'* is the branch where you want your changes to go, ie, your classmate's repository. *'head branch'* is the branch where you made *your* changes. Make sure these are set properly. Remember: the first one is the TO, the second one is the FROM: the place where you want your changes to go TO, FROM the place where you made the changes.

f. A pull request has to have a message attached to it, so that your classmate knows what kind of change you're proposing. Fill in the message fields appropriately, then hit the 'create pull request' button.

8. Finally, the last bit of work to be done is to accept the pull request and `merge` the changes into the original repository.

a. Go to your repository on your Github account. Check to see if there are any 'pull requests' - these will be listed under the 'pull requests' tab. Click on that tab.

b. You can merge from the command line, but for now, you can simply click on the green 'merge pull request' button, and then the 'confirm merge' button. The changes your classmate has made have now been folded into your repository.

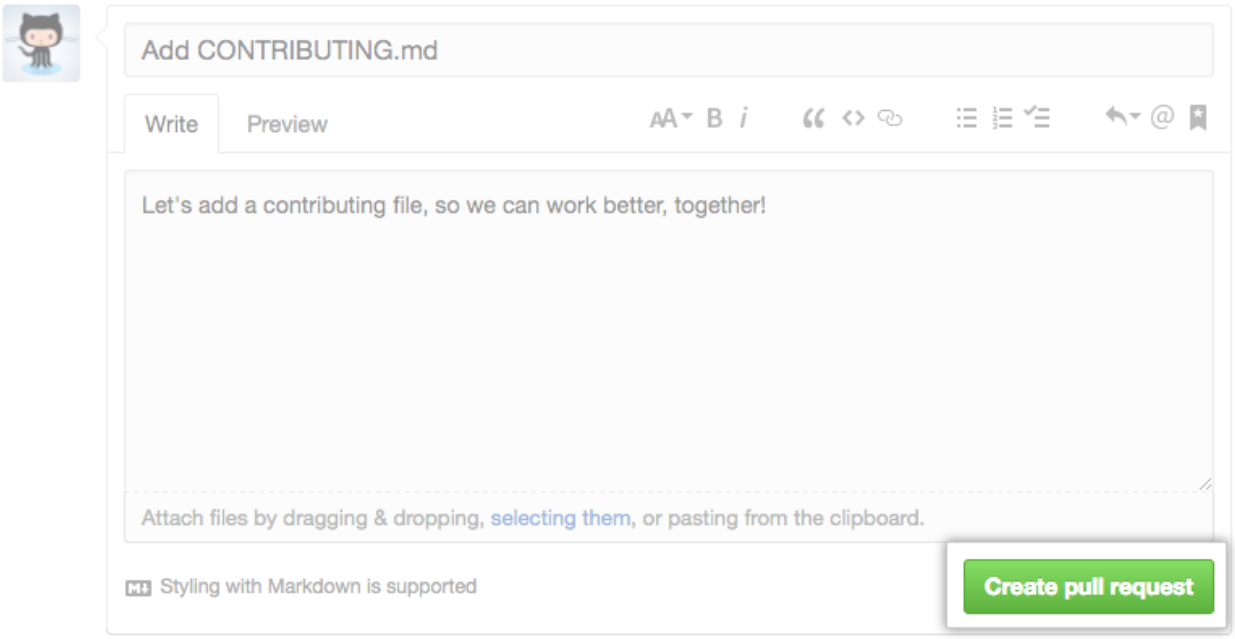

Figure 1.4: A pull request message; Image from Github.com

c. To get the updates on your local machine, go back to the command line and type

```
$ git pull origin master
```

1.3.6 Warnings

It is possible to make changes to files directly via the edit button on Github. Be careful if you do this, because things rapidly can become out of sync, resulting in conflicts between differing versions of the same file. Get in the habit of making your changes on your own machine, and making sure things are committed and up-to-date (`git status`, `git pull origin master`, `git fetch upstream` are your friends) before beginning work. At this point, you might want to investigate some of the graphical interfaces for Git (such as Github Desktop). Knowing as you do how things work from the command line, the idiosyncracies of the graphical interfaces will make more sense. For further practice on the ins-and-outs of Git and Github Desktop, we recommend trying the Git-it app by Jessica Lord.

For help in resolving merge conflicts, see the Github help documentation. For a quick reminder of how the workflow should go, see this cheat-sheet by Chase Pettit.

1.4 Open Notebook Research & Scholarly Communication

Digital archaeology necessarily generates a lot of files. Many of those files are data; many more are manipulations of that data, or the data in various stages of cleaning and analysis. Without any sort of version control or revision history (as detailed in the previous section), these files quickly replicate to the point where a project can be in serious danger of failing. Which file contains the 'correct' data? The correct analysis? Even worse, imagine coming back to a project after a few months' absence. Worse still, after a major operating system update of the kind foisted on Windows users from Windows 7 to Windows 10. The bad news continues: magnetic storage can fail; online cloud services can be hacked; a key person on the project can die.

Even if the data makes it to publication, there is the problem of the data not being available to others for re-interrogation or re-analysis. Requests for data from the authors of journal articles are routinely ignored, whether by accident or design. Researchers may sit on data for years. We have all of us had the experience of working on a collection of material, and then writing to the author of the original article, requesting an explanation for some aspect of the data schema used, only to find out that the author has either died, kept no notes, left the field entirely, or simply doesn't remember.

There is no excuse for this any longer. `Open notebook science` is a gathering movement across a number of fields to make the entire research process transparent by sharing materials online as they are generated. These include everything from the data files themselves, to the code used to manipulated it, to notes and observations in various archives. Variations on this 'strong' position include data-publishing of the materials after the main paper has been published (see for instance OpenContext or the Journal of Open Archadological Data). Researchers such as Ben Marwick and Mark E. Madsen are leading the field in archaeology, while scholars such as Caleb McDaniel are pushing the boundaries in history. The combination of simple text files (whether written text or tabular data such as .csv files) with static website generators (ie, html rather than dynamically generated database websites like Wordpress) enables the live publishing of in-progress work. Carl Boettiger is often cited as one of the godfathers of this movement. He makes an important distinction:

> This [notebook, not blog] is the active, permanent record of my scientific research, standing in place of the traditional paper bound lab notebook. The notebook is primarily a tool for me to do science, not communicate it. I write my entries with the hope that they are intelligible to my future self; and maybe my collaborators and experts in my field. Only the occasional entry will be written for a more general audience. [...] In these pages you will find not only thoughts and ideas, but references to the literature I read, the codes or manuscripts I write, derivations I scribble and graphs I create and mistakes I make. (Boettiger)

Major funding bodies are starting to require a similar transparency in the research that they support. Recently, the Social Sciences and Humanities Research Council of Canada published guidance on data management plans:

> All research data collected with the use of SSHRC funds must be preserved and made available for use by others within a reasonable period of time. SSHRC considers "a reasonable period" to be within two years of the completion of the research project for which the data was collected.

Annecdotally, we have also heard of work being denied funding because the data management plan, and/or the plan for knowledge mobilization, made only the briefest of nods towards these issues: 'we shall have a blog and will save the data onto a usb stick' does not cut it any more. A recent volume of case-studies in 'reproducible research' includes a contribution from Ben Marwick that details not only the benefits of such an approach, but also the 'pain points'. Key amongst them was that not everyone participating in the project was on board using scripted code to perform the analysis (preferring instead to use the point-and-click of Excel), the duplication of effort that emerged as a result, and the complexities that arose from what he calls the 'dual universes' of Microsoft tools versus the open source tools. (MARWICK REF). On the other hand, the advantages outweighed the pain. For Marwick's team, because their results and analysis can be re-queried and re-interrogated, they have an unusually high degree of confidence in what they've produced. Their data, and their results have a complete history of revisions that can be examined by reviewers. Their code can be re-used and re-purposed, thus making their subsequent research more efficient. Marwick goes on to create an entire 'compendium' of code, notes, data, and software dependencies that can be duplicated by other researchers. Indeed, we will be re-visiting their compendium in Section XXXXXXXXX.

Ultimately, McDaniels says it best about keeping open notebooks of research in progress when he writes,

> The truth is that we often don't realize the value of what we have until someone else sees it. By inviting others to see our work in progress, we also open new avenues of interpretation, uncover new linkages between things we would otherwise have persisted in seeing as unconnected, and create new opportunities for collaboration with fellow travelers. These things might still happen through the sharing of our notebooks after publication, but imagine how our publications might be enriched and improved if we lifted our gems to the sunlight before we decided which ones to

set and which ones to discard? What new flashes in the pan might we find if we sifted through our sources in the company of others?

A parallel development is the growing practice of placing materials online as pre-prints or even as drafts, for sharing and for soliciting comments. Graham for instance uses a blog as a place to share longer-form discursive writing in progress; with his collaborators Ian Milligan and Scott Weingart, he even wrote a book 'live' on the web, warts and all (which you may still view at The Macroscope). Sharing the draft in progress allowed them to identify errors and ommissions as they wrote, and for their individual chapters and sections to be incorporated into class syllabi right away. In their particular case, they came to an arrangment with their publisher to permit the draft to remain online even after the formal publication of the 'finished' book - which was fortunate, as they ended up writing another chapter immediately after publication! In this, they were building on the work of scholars such as Kathleen Fitzpatrick, whose *Planned Obsolescence* was one of the first to use the Media Commons 'comment press' website to support the writing. Commentpress is a plugin for the widely used Wordpress blogging system, which allows comments to be made at the level of individual paragraphs. This textbook you are currently reading uses another solution, the hypothes.is plugin that fosters communal reading and annotation of electronic texts. This points to another happy by-product of sharing one's work this way - the ability to generate communities of interest around one's research. The Kitz et al. volume is written with the Gitbook platform, which is a graphical interface for writing using Git at its core with markdown text files to manage the collaboration. The commit history for the book then also is a record of how the book evolved, and who did what to it when. In a way, it functions a bit like 'track changes' in Word, with the significant difference that the evolution of the book can be rewound and taken down different branches when desired.

In an ideal world, we would recommend that everyone should push for such radical transparency in their research and teaching. But what is safe for a group of (mostly) white, tenured, men is not safe for everyone online. In which case, what we recommend is for individuals to assess what is safest for them to do, while still making use of the affordances of Git, remote repositories, and simple text files. Bitbucket at the time of writing offers free private repositories (so you can push your changes to a remote repository without fear of others looking or cloning your materials); ReclaimHosting supports academic webhosting and allows one to set up the private 'dropbox' like file-sharing service Owncloud.

- in this section, discuss people like caleb mcdaniels, lincoln mullen, ben marwick, other archaeologists,
- develop exercise to use lincoln's framework for simple notebooks, Ben's examples for complex notebooks
- discuss the role of blogging in this, and how a blog doesn't necessarily have to be reflective or narrative, brings back to original roots of the term, web log.
- discuss the dangers inherent in doing this kind of thing in the open, especially given hot mess that the web is
- be as open as it is safe for you to be. one could set up an owncloud via reclaim hosting, and push materials there in both open and closed versions: here at least you've got LOCKSS
- Ben Marwick's demo repos for archae projects https://github.com/benmarwick/researchcompendium
- reproducibility case studies project: https://github.com/benmarwick/repro-case-studies
- https://www.practicereproducibleresearch.org/case-studies/benmarwick.html
- the dangers of excel

1.4.1 discussion

1.4.2 exercises

1.5 Failing Productively

- video from msudai ?

1.5.1 discussion

1.6 Introduction to Digital Libraries, Archives & Repositories

yadda linked open data in here?

1.7 Command Line Methods for Working with APIs

yadda ... maybe do this in R (since we have an Rserver in the box) and use the rcats tutorial as a model? https://rforcats.net/ which might fit into the section below

1.7.1 Working with Open Context

Open Context, http://opencontext.org operates under the idea that every element of an archaeological research project should be published. To that end, they publish *everything* with its own unique URI.

Search for something interesting. I put 'poggio' in the search box, and then clicked on the various options to get the architectural fragments. Look at the URL: https://opencontext.org/subjects-search/?prop=oc-gen-cat-object&q=Poggio#15/43.1526/11.4090/19/any/Google-Satellite See all that stuff after the word 'Poggio'? That's to generate the map view. We don't need it.

We're going to ask for the search results w/o all of the website extras, no maps, no shiny interface. To do that, we take advantage of the API. With open context, if you have a search with a '?' in the URL, you can put .json in front of the question mark, and delete all of the stuff from the # sign on, like so:

https://opencontext.org/subjects-search/.json?prop=oc-gen-cat-object&q=Poggio

Put that in the address bar. Boom! lots of stuff! But only one page's worth, which isn't lots of data. To get a lot more data, we have to add another parameter, the number of rows: ?rows=100&. Slot that in just before the p in prop= and see what happens.

Now, that isn't all of the records though. Remove the .json and see what happens when you click on the arrows to page through the NEXT 100 rows. You get a URL like this:

https://opencontext.org/subjects-search/?rows=100&prop=oc-gen-cat-object&start=100&q=Poggio#15/43.1526/11.4090/19/any/Google-Satellite

So – to recap, the URL is searching for 100 rows at a time, in the general object category, starting from row 100, and grabbing materials from Poggio. We now know enough about how open context's api works to grab material.

Couple of ways one could grab it:

1. You could copy n' paste -> but that will only get you one page's worth of data (and if you tried to put, say, 10791 into the 'rows' parameter, you'll just get a time-out error). You'd have to go back to the search page, hit the 'next' button, reinsert the .json etc over and over again.
2. Automatically. We'll use a program called wget to do this. (To install wget on your machine, see the programming historian Wget will interact with the Open Context site to retrieve the data. We feed wget a file that contains all of the urls that we wish to grab, and it saves all of the data into a single file. So, open a new text file and paste our search URL in there like so:

```
https://opencontext.org/subjects-search/.json?rows=100&prop=oc-gen-cat-object---oc-gen-cat-arch-element&
https://opencontext.org/subjects-search/.json?rows=100&prop=oc-gen-cat-object---oc-gen-cat-arch-element&
https://opencontext.org/subjects-search/.json?rows=100&prop=oc-gen-cat-object---oc-gen-cat-arch-element&
```

. . . and so on until we've covered the full 4000 objects. Tedious? You bet. So we'll get the computer to generate those URLS for us. Open a new text file, and copy the following in:

```
#URL-Generator.py

urls = '';
f=open('urls.txt','w')
for x in range(1, 4000, 100):
    urls = 'https://opencontext.org/subjects-search/.json?rows=100&prop=oc-gen-cat-object---oc-gen-cat-a
    f.write(urls)
f.close
```

and save it as url-generator.py. This program is in the python language. Type at the prompt:

```
$ python url-generator.py
```

This little program defines an empty container called 'urls'; it then creates a new file called 'urls.txt'; then we tell it to write the address of our search into the urls container. See the %d in there? The program writes a number between 1 and 4000; each time it does that, it counts by 100 so that the next time it goes through the loop, it adds a new address with the correct starting point! Then it saves that container of URLs into the file urls.txt. Go ahead, open it up, and you'll see.

Now we'll feed it to wget like so. At the prompt type

```
$ wget -i urls.txt -r --no-parent -nd -w 2 --limit-rate=10k
```

You'll end up with a lot of files that have no file extension in your folder, eg

```
.json?rows=100&prop=oc-gen-cat-object---oc-gen-cat-arch-element&start=61&q=Poggio%2F
```

Rename them so that they have `.json` file extensions. Now we concatenate them together

```
# As simple as this. Output file should be last
$ json-concat file1.json file2.json file3.json file4.json ouput.json
```

SG MAKE SURE THAT JSON-CONCAT CAN BE HAD ON DHBOX, OTHERWISE FIND ALTERNATIVE

ALSO MAKE SURE TO TALK THROUGH THIS: https://github.com/ropensci/opencontext

json is a text file where keys are paired with values. JQ is a piece of software that enables us to reach into a json file, grab the data we want, and create either new json or csv. If you intend to visualize and explore data using some sort of spreadsheet program, then you'll need to extract the data you want into a csv that your spreadsheet can digest. If you wanted to try something like d3 or some other dynamic library for generating web-based visualizations (eg p5js), you'll need json.

jqplay

JQ lets us do some fun filtering and parsing, but we won't download and install it yet. Instead, we'll load some sample data into a web-toy called jqplay. This will let us try different ideas out and see the results immediately. In the this file called sample.json I have the query results from Open Context – Github recognizes that it is json and that it has geographic data within it, and turns it automatically into a map! To see the raw json, click on the < > button. Copy that data into the json box at jqplay.org.

JQPlay will colour-code the json. Everything in red is a key, everything in black is a value. Keys can be nested, as represented by the indentation. Scroll down through the json – do you see any interesting key:value pairs? Matthew Lincoln's tutorial at the programming historian is one of the most cogent explanations of how this works, and I do recommend you read that piece. Suffice to say, for now, that if you see an interesting key:value pair that you'd like to extract, you need to figure out just how deeply nested it is. For instance, there is a properties key that seems to have interesting information within it about dates, wares, contexts and so on. Perhaps we'd like to build a query using JQ that extracts that information into a csv. It's within the features key pair, so try entering the following in the filter box:

```
.features [ ] | .properties
```

You should get something like this:


```
{
  "id": "#geo-disc-tile-12023202222130313322",
  "href": "https://opencontext.org/search/?disc-geotile=120232022222130313322&prop=oc-gen-cat-object&rows
  "label": "Discovery region (1)",
  "feature-type": "discovery region (facet)",
  "count": 12,
  "early bce/ce": -700,
  "late bce/ce": -535
}
{
  "id": "#geo-disc-tile-12023202222130313323",
  "href": "https://opencontext.org/search/?disc-geotile=120232022222130313323&prop=oc-gen-cat-object&rows
  "label": "Discovery region (2)",
  "feature-type": "discovery region (facet)",
  "count": 25,
  "early bce/ce": -700,
  "late bce/ce": -535
}
```


For the exact syntax of why that works, see Lincoln's tutorial. I'm going to just jump to the conclusion now. Let's say we wanted to grab some of those keys within properties, and turn into a csv. We tell it to look inside features and find properties; then we tell it to make a new array with just those keys within properties we want; and then we tell it to pipe that information into comma-separated values. Try the following on the sample data:

```
.features [ ] | .properties | [.label, .href, ."context label", ."early bce/ce", ."late bce/ce", ."item
```

. . . and make sure to tick the 'raw output' box at the top right. Ta da! You've culled the information of interest from a json file, into a csv. There's a lot more you can do with jq, but this will get you started. Finally, we move back to the command line and invoke JQ to format the data how we want it:

```
jq -r '.features [ ] | .properties | [.label, .href, ."context label", ."early bce/ce", ."late bce/ce",
```

1.7.2 Working with Omeka

yadda

1.7.3 Working with tDAR

yadda

1.7.4 Working with ADS

1.7.5 Exercises

yadda

1.8 The Ethics of Big Data in Archaeology

Ethics! Lots of Ethics! - include discussion of privacy - ethics of crowdfunding - ethics of crowdsourcing (& dubious ethics of using amazon's mechanical turk) - ethics of public archaeology & social media

1.8.1 discussion

1.8.2 exercises

- maybe some sort of personal audit of what traces the student is leaving online
- some sort of case study?

Chapter 2

Making Data Useful

blah blah introd

2.1 Designing Data Collection

yada yada

2.1.1 discussion

2.1.2 exercises

2.2 Cleaning Data with Open Refine

blahde blah blah

2.2.1 discussion

2.2.2 exercises

2.3 Linked Open Data and Data Publishing

yargble blarble floss

2.3.1 discussion

2.3.2 exercises

Chapter 3

Finding and Communicating the Compelling Story

blah blah blah

3.1 Statistical Computing with R and Python Notebooks; Reproducible code

blah

3.1.1 discussion

3.1.2 exercises

3.2 D3, Processing, and Data Driven Documents

blerg

3.2.1 discussion

3.2.2 exercises

3.3 Storytelling and the Archaeological CMS: Omeka, Kora

blargle

3.3.1 Omeka

bla

3.3.2 Kora

3.3.3 Exercises

3.4 Web Mapping with Leaflet

...I wonder if we should talk about GIS & Pandas, etc... or in R?

3.4.1 discussion

3.4.2 exercises

3.5 Place-based Interpretation with Locative Augmented Reality

yep.

3.5.1 discussion

3.5.2 exercises

3.6 Archaeogaming and Virtual Archaeology

yay archaeogaming

3.6.1 discussion

3.6.2 exercises

3.7 Social media as Public Engagement & Scholarly Communication in Archaeology

boo socmed

3.7.1 discussion

3.7.2 exercises

Chapter 4

Eliding the Digital and the Physical

crazytown

4.1 3D Photogrammetry & Structure from Motion

vsfm

4.1.1 discussion

4.1.2 exercises

4.2 3D Printing, the Internet of Things and "Maker" Archaeology

yay

4.2.1 discussion

4.2.2 exercises

4.3 Artificial Intelligence in Digital Archaeology

4.3.1 agent models

blah

4.3.2 discussion

blah

4.3.3 exercises

blah

4.3.4 machine learning for image captioning and other classificatory tasks

blah

4.3.5 discussion

blah

4.3.6 exercises

Chapter 5

Digital Archaeology's Place in the World

blerg

5.1 Marketing Digital Archaeology

blag

5.1.1 discussion

5.1.2 exercises

5.2 Sustainability & Power in Digital Archaeology

the big ticket item.

5.2.1 discussion

5.2.2 exercises

Chapter 6

On the Horizons: Where Digital Archaeology Might Go Next

blargble

References

Boettiger, Carl"Welcome to My Lab Notebook - Reloaded." Website. *Lab Notebook*. http://www.carlboettiger.info/09/28/Welcome-to-my-lab-notebook.html.

Caraher, William. 2012. "Archaeological Glitch Art." *The Archaeology of the Mediterranean World*. https://mediterraneanworld.wordpress.com/2012/11/21/archaeological-glitch-art/.

Evans, Thomas Laurence, and Patrick Daly, eds. 2006. *Digital Archaeology: Bridging Method and Theory*. Psychology Press.

Goldstone, Andrew. 2018. "Teaching Quantitative Methods: What Makes It Hard 9in Literary Studies)." In *Debates in the Digital Humanities*.

Graham, Shawn. 2017. "Cacophony: Bad Algorithmic Music to Muse To." https://electricarchaeology.ca/2017/02/03/cacophony-bad-algorithmic-music-to-muse-to/.

Liu, Alan. 2004. *The Laws of Cool: Knowledge Work and the Culture of Information*. 1 edition. Chicago: University of Chicago Press.

Montello, Daniel R., Sara Irina Fabrikant, Marco Ruocco, and Richard S. Middleton. 2003. "Testing the First Law of Cognitive Geography on Point-Display Spatializations." In *International Conference on Spatial Information Theory*, 316–31. Springer. http://link.springer.com/chapter/10.1007/978-3-540-39923-0_21.

Mullen, Lincoln. 2017. "A Confirmation of Andrew Goldstone on 'Teaching Quantitative Methods'." *The Backward Glance*. http://lincolnmullen.com/blog/a-confirmation-of-andrew-goldstone-on-teaching-quantitative-methods/.

Owens, Trevor. 2012. "Discovery and Justification Are Different: Notes on Science-Ing the Humanities." *Trevor Owens*. http://www.trevorowens.org/2012/11/discovery-and-justification-are-different-notes-on-sciencing-the-humanities/.

Ramsay, Stephen. 2011. *Reading Machines: Toward an Algorithmic Criticism*. 1st Edition edition. Urbana: University of Illinois Press.

Samuels, Lisa, and Jerome J. McGann. 1999. "Deformance and Interpretation." *New Literary History* 30 (1): 25–56. doi:10.1353/nlh.1999.0010.