Hybrid workflow how-to: Making automated workflows, part 1

As part of the [INC subgroup](http://networkcultures.org), we have been developing a workflow that allows a flexible production of different kinds of electronic outputs like EPUB, PDF, and [book trailers](http://digitalpublishingtoolkit.org/2014/10/epub-trailers/) from a sample collection of essays from the recently published [Society of the Query](http://networkcultures.org/query/) Reader.

# Part 1

In part one of this tutorial, we look at using the [pandoc](http://johnmacfarlane.net/pandoc/) tool on the command line to convert a [markdown](http://daringfireball.net/projects/markdown/) source that has been edited by an editor into HTML and EPUB outputs. In addition, we will add metadata and use pandoc templates and a stylesheet to customize the output.

This tutorial is targeted for developers or people interested in creating automated workflows for producing EPUBs. It assumes basic familiarity with a commandline interface (such as the Terminal application on GNU/Linux or Mac OS X, or the command prompt in Windows). Introductions to the commandline such as the [Designers guide to the OS X Command Prompt](http://wiseheartdesign.com/articles/2010/11/12/the-designers-guide-to-the-osx-command-prompt/) can be very useful if the commandline is still new to you.

## Install pandoc

Instructions for installing pandoc on Mac, Windows, and Linux are given on the [pandoc website](http://johnmacfarlane.net/pandoc/installing.html).

Mac: From the [download page](https://github.com/jgm/pandoc/releases), find the green button with a link that ends with "osx.pkg". Download and install this.

Debian/Ubuntu: Pandoc is available from your package manager:

sudo apt-get install pandoc

However, the version of pandoc is typically outdated. To compile the latest and greatest, follow the instructions on the [pandoc website](http://johnmacfarlane.net/pandoc/installing.html) under "All platforms". In a nutshell:

sudo apt-get install haskell-platform  
cabal update  
cabal install pandoc

## Prepare your workspace & tools

Unpack, checkout or copy the sample files from the [github repository](https://github.com/DigitalPublishingToolkit/Society-of-the-Query-Reader). Open the Terminal and use the cd command to enter the "part1" folder in the developer section of the how-to-tutorial files.

git clone https://github.com/DigitalPublishingToolkit/Society-of-the-Query-Reader.git  
  
cd how-to-tutorial/developer/part1

As we will be working with different kinds of code (both markdown and HTML output), it is quite important to have a good code editor to work with. A code editor (as opposed to programs like "Simple Text" or Microsoft Word) provides features like language-specific syntax coloring and helpful keyboard shortcuts for say adding comments. Many code editors exist from classical commandline editors like vi and emacs, to graphical editors like gedit and [Sublime Text](http://www.sublimetext.com/). Though not Free Software, [Sublime Text](http://www.sublimetext.com/) is an excellent cross-platform code editor with many advanced features, which is free of charge to use (in Trial mode) indefinitely and a good example of the state of the art of contemporary code editors in terms of features and usability.

## Use pandoc to convert a single markdown source file to HTML & EPUB

As you may have seen in Step 5 of the [Hybrid Workflow Howto for Editors](http://digitalpublishingtoolkit.org/2014/10/hybrid-workflow-how-to-introduction-editing-steps/), you can use the pandoc program to convert from one file format to another (such as in that example from Word DOCX to Markdown). In this step we use pandoc to convert an edited markdown file as it would arrive from an editor into first HTML, and then to an EPUB.

Open the file Kylie-Jarett.md in your editor, it begins:

# A Database of Intention?  
  
Kylie Jarrett  
  
In his 2005 study of Google, industry analyst John Battelle describes  
the company’s technology as a ‘database of intentions’, ‘a massive  
clickstream database of desires, needs, wants, and preferences that can  
be discovered, subpoenaed, archived, tracked, and exploited for all  
sorts of ends’.[^1]

In the terminal type the following. Note that once you type the K of the filename you should be able to press the tab key to "auto-complete" the name of the file:

pandoc Kylie-Jarrett.md

By default pandoc will attempt to guess the type of the input file based on the file extension. In this case the ".md" means that pandoc assumes Markdown input. By default pandoc produces HTML and prints it to the terminal rather than saving it in a file. To save it to a file, you can use pandoc's -o option:

pandoc Kylie-Jarrett.md -o test.html

You can also explicitly state input and output types with pandoc's --from and --to options. This can be useful if a filename misses a recognizable extension:

pandoc --from markdown --to html Kylie-Jarrett.md -o test.html

In general the order of the parameter doesn't really matter, as long as the options precede their values, so the following would be the same:

pandoc -o test.html --to html --from markdown Kylie-Jarrett.md

Use a web browser to open the resulting file (test.html) and check the output. It should appear as formatted HTML. However there are likely some glitches in the text. This is because pandoc's default HTML output is merely a fragment, and not a complete HTML document, and some information (such as the proper encoding of the text) is not included. If you open test.html in your editor, you see that the file begins:

<h1 id="a-database-of-intention">A Database of Intention?</h1>  
<p>Kylie Jarrett</p>  
...

Note that pandoc automatically assigns an id to the header. This is useful when linking. Next, run pandoc again, this time adding the --standalone (or -s) option:

pandoc essays/Kylie-Jarrett.md --standalone -o test.html

If you reload test.html in the browser, you should see that the character gliches are corrected. If you look at test.html in your editor, you see it now begins:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
 <meta http-equiv="Content-Style-Type" content="text/css" />  
 <meta name="generator" content="pandoc" />  
 <title></title>  
 <style type="text/css">code{white-space: pre;}</style>  
</head>  
<body>  
<h1 id="a-database-of-intention">A Database of Intention?</h1>  
...

The output now contains a proper HTML doctype and head section with, among other things a character set, which tells the browser to interpret the text as encoded in the utf-8 standard (Browsers by default assume the latin-1 character set when a document doesn't state it's encoding which is why the characters were being misinterpreted in the fragment).

## Add metadata to your document

In the HTML output in the previous step, the title tag in the document is left blank. Even though the title of the essay is in the document, it's a level one header, pandoc doesn't make any assumptions that that is a title. Pandoc supports adding "metadata" (data about the document itself).

Add the following to the first lines of the file "essays/Kylie\_Jarrett.md":

---  
title: A Database of Intention?  
author: Kylie Jarrett  
---

Now repeat the pandoc command to update the test output:

pandoc essays/Kylie-Jarrett.md --standalone -o test.html

The resulting document now looks like this:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
 <meta http-equiv="Content-Style-Type" content="text/css" />  
 <meta name="generator" content="pandoc" />  
 <meta name="author" content="Kylie Jarrett" />  
 <title>A Database of Intention?</title>  
 <style type="text/css">code{white-space: pre;}</style>  
</head>  
<body>  
<div id="header">  
<h1 class="title">A Database of Intention?</h1>  
<h2 class="author">Kylie Jarrett</h2>  
</div>  
<h1 id="a-database-of-intention">A Database of Intention?</h1>  
<p>Kylie Jarrett</p>  
<p>In his 2005 study of Google, industry analyst John Battelle describes the

Pandoc detects the title and author settings in the metadata and automatically adds them to both the HTML header (as the title and a meta tag), as well as creating an h1 and h2 tag at the start of the document. The problem now is that the title and author's name appears twice. It's best to remove the title and author's name form the body of the document.

Remove the original title & author lines from Kylie-Jarrett.md so that it looks like this:

---  
title: A Database of Intention?  
author: Kylie Jarrett  
---  
  
In his 2005 study of Google, industry analyst John Battelle describes  
the company’s technology as a ‘database of intentions’, ‘a massive  
clickstream database of desires, needs, wants, and preferences that can  
be discovered, subpoenaed, archived, tracked, and exploited for all  
sorts of ends’.[^1]

And once again:

pandoc essays/Kylie-Jarrett.md --standalone -o test.html

This should look better. In the next step we see how to use pandoc's template feature to customize how the metadata is displayed in the output. Using metadata in this way is a useful way to create a more flexible source document that can produce different kinds of outputs and tailored to different contexts.

The format pandoc uses for metadata is called [YAML](http://www.yaml.org/). One important detail to note is that if a title includes a colon character (:), you need to put quotation marks around the title, as in:

---  
title: "Educating for Search: Understanding the Past and Present Search Technology to Teach for Future Resilience"  
author: Dave Crusoe  
---

To give multiple authors, you can use a list by using square brackets and separating names with commas, as in:

---  
title: "Polluted and Predictive, in 133 Words"  
author: [Mél Hogan, M.E. Luka]  
---

When converted to HTML produces:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
 <meta http-equiv="Content-Style-Type" content="text/css" />  
 <meta name="generator" content="pandoc" />  
 <meta name="author" content="Mél Hogan" />  
 <meta name="author" content="M.E. Luka" />  
 <title>Polluted and Predictive, in 133 Words</title>  
 <style type="text/css">code{white-space: pre;}</style>  
</head>

## Customize your output with a template and a stylesheet

So how did pandoc know what to do with the title and author in your metadata? It turns out that [pandoc has a collection of standard templates](http://johnmacfarlane.net/pandoc/README.html#templates), or for each output format, which it uses to produce its output.

To see pandoc's template for producing HTML output, type the command:

pandoc -D html

The output (in part) shows:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml"$if(lang)$ lang="$lang$" xml:lang="$lang$"$endif$>  
<head>  
 ...  
 <meta name="author" content="$author-meta$" />  
$endfor$  
$if(date-meta)$  
 <meta name="date" content="$date-meta$" />  
$endif$  
 <title>$if(title-prefix)$$title-prefix$ - $endif$$pagetitle$</title>  
 ...  
$for(css)$  
 <link rel="stylesheet" href="$css$" $if(html5)$$else$type="text/css" $endif$/>  
$endfor$  
</head>  
<body>  
...  
$if(title)$  
<div id="$idprefix$header">  
<h1 class="title">$title$</h1>  
$if(subtitle)$  
<h1 class="subtitle">$subtitle$</h1>  
$endif$  
$for(author)$  
<h2 class="author">$author$</h2>  
$endfor$  
$if(date)$  
<h3 class="date">$date$</h3>  
$endif$  
</div>  
$endif$

In the template, you can see that pandoc provides some sophisticated tools like conditionals (if) and loops (for) to provide basic handling for optional elements and lists author names. To customize this standard template, make a copy of it named custom.html:

pandoc -D html > custom.html

Open the custom.html and change the display of the title and author to:

$if(title)$  
<div id="$idprefix$header">  
 <h1 class="title">$title$</h1>  
 <div class="authors">  
 <div class="separator">&#172;</div>  
 $for(author)$  
 <div class="author">$author$</div>  
 $endfor$  
 </div>  
</div>  
$endif$

You can also see in the template that pandoc provides a number of ways of adding custom stylesheets. The easiest is to use the --css option. So create a new file names "styles.css" with the following:

body {  
 font-family: sans-serif;  
}  
#header {  
 text-align: center;  
}  
h1 {  
 margin-bottom: 0;  
}  
.author {  
 font-weight: bold;  
}

And now bring it all together with:

pandoc Kylie-Jarrett.md --standalone --template custom.html --css styles.css -o Kylie-Jarrett.html

## Produce an EPUB

You can easily [produce an EPUB from a markdown source with pandoc](http://johnmacfarlane.net/pandoc/epub.html) by simply specifying an EPUB extension to the output file. Note that the --standalone option is implicit with an EPUB:

pandoc Kylie-Jarrett.md -o Kylie-Jarrett.epub

To specify custom CSS with an EPUB, use the --epub-stylesheet option:

pandoc Kylie-Jarrett.md --epub-stylesheet styles.css -o Kylie-Jarrett.epub

Note that pandoc places an automatically generated table of contents as the last page, to move this to the front use the --table-of-contents option.

pandoc Kylie-Jarrett.md --epub-stylesheet styles.css -o Kylie-Jarrett.epub --table-of-contents

# Part 2

In part two of this tutorial, we create a shell script to compile multiple [markdown](http://daringfireball.net/projects/markdown/) sources into a EPUB-format Reader using [pandoc](http://johnmacfarlane.net/pandoc/). We then use a helper script, expand-toc.py, to use a markdown-formatted table of contents to order the contents of the EPUB. Finally, we create a [makefile](http://digitalpublishingtoolkit.org/2014/10/make-book/) to fully automate the build process, and add an [EPUB trailer](http://digitalpublishingtoolkit.org/2014/10/epub-trailers/) as an output.

This tutorial is targeted for developers or people interested in creating automated workflows for producing EPUBs. It assumes basic familiarity with a commandline interface (such as the Terminal application on GNU/Linux or Mac OS X, or the command prompt in Windows).

## Prepare your workspace & tools

Unpack, checkout or copy the sample files from the [github repository](https://github.com/DigitalPublishingToolkit/Society-of-the-Query-Reader). Open the Terminal and use the cd command to enter the "part2" folder in the developer section of the how-to-tutorial files.

git clone https://github.com/DigitalPublishingToolkit/Society-of-the-Query-Reader.git  
  
cd how-to-tutorial/developer/part2

## Use pandoc to tweak a markdown document to be combinable with others

Any filename given to pandoc on the commandline that is not preceded by an option (such as -o) is considered an input. When you give pandoc multiple input files, pandoc cuts and pastes the different texts together as if they were coming from a single file. While this will sometimes just work, it creates some specific problems for the essays from the Society of the Query:

pandoc -o reader.epub \*.md

Produces a lot of warnings:

pandoc: Duplicate note reference `3' "source" (line 1582, column 1)  
pandoc: Duplicate note reference `2' "source" (line 1580, column 1)  
pandoc: Duplicate note reference `1' "source" (line 1578, column 1)

This is because when treated as a single document, all the footnotes (which use numeric indexes) from the different essays can no longer be differentiated (reference 3 from which essay?).

Besides overlapping footnotes, there may be problems with the hierarchy of the overall document. The individudal essays have been written with their title as level 1, and sections within the essay as level 2. This is fine, but for the final reader we may want to introduce higher level sections (at level one) to for instance group the essays thematically. In that case the essay levels should shift so that their title is level 2, and essay sections level 3. Luckily pandoc has an option to do just that called --base-header-level that (re)sets the "topmost" level of a document when outputting it.

In general, a useful technique is to use pandoc in two passes, first individually on each essay to adjust it for compilation in the larger document. Each altered essay is then pasted in a new master markdown file containing the entire reader document and it is this file that is converted (again by pandoc) to EPUB.

Try the following command:

pandoc David\_Crusoe.md --to markdown --id-prefix=David\_Crusoe.md- --base-header-level 2

In the resulting markdown output, note the final Notes section:

### Notes {#notes .notes}  
  
[^David\_Crusoe.md-1]: The Common Core references algebraic set theory in  
 its high school appendices. See  
 http://www.corestandards.org/assets/CCSSI\\_Mathematics\\_Appendix\\_A.pdf  
 for more information.  
  
[^David\_Crusoe.md-2]: Way Back Machine, 21 November 1996 archive of ‘How  
 to Use Excite Search’,  
 http://web.archive.org/web/19961219003220/http://www.excite.com/Info/advanced.html?aqt.  
  
[^David\_Crusoe.md-3]: Anthony Stuart, ‘Re: Boolean + Operator Removed?  
 Why?’ posting to Google Search Forum, 5 November 2011,  
 http://productforums.google.com/forum/\#!searchin/websearch/%22implied\$20and%22/websearch/3oIWbew9xdE/xuKBfNk5wjwJ.

First the "Notes" header has shifted from a level 2 to level 3. The "--base-header-level 2" option basically shifts each level by one (in other words what was 1 becomes 2, was 2 becomes 3 and so on).

Additionally, the footnote references now are preceded by the name of the document (and pandoc has changed the matching references in the text as well). This markdown will now combine well with other sources.

Next, let's use a template (see essay-template01.md) to add a custom header and format the authors names:

# $title$ {.title}  
  
<div class="separator">&#172;</div>  
$for(author)$  
<div class="author">$author$</div>  
$endfor$  
  
$body$  
$for(include-after)$  
$include-after$  
$endfor$

Now putting it all together:

pandoc David\_Crusoe.md --to markdown --template essay-template-01.md --no-wrap --id-prefix=David\_Crusoe.md- --base-header-level 2

*Devilish detail*: Note the use of the --no-wrap option. This disables pandoc's automatic text wrapping feature when outputting, in this case, markdown. A subtle bug we discovered was that long titles were being line wrapped which would then break the h2 header in the template (where the variable is expanded on a line starting with ## so that only the first line was considered part of the header). Thankfully this behaviour can be disabled with --no-wrap.

## Use pandoc to convert multiple markdown source files to an EPUB

Open the file build01.sh, it contains the following:

#!/bin/bash  
  
# blank the reader.md file  
> reader.md  
  
# process all the files listed in ESSAYS in a loop  
# use >> to append each output to reader.md  
for i in `cat ESSAYS`  
do  
pandoc "$i" \  
 --to markdown \  
 --template essay-template-01.md \  
 --no-wrap \  
 --id-prefix="${i}-" \  
 >> reader.md  
  
# ensure a blank line at the end of each essay  
echo >> reader.md  
  
done  
  
# Make the EPUB from reader.md  
pandoc \  
 --epub-chapter-level=1 \  
 --toc-depth=1 \  
 -o reader.epub \  
 reader.md

Run the build01 script by typing:

bash build01.sh

*Devilish detail*: When combining all the markdown into a single document, all the footnotes would be merged together and displayed in one large section at the end of the EPUB. A crucial pandoc feature is the --epub-chapter-level option which in addition to splitting the source (back) into separate files (which makes the document lighter for an e-reader), it also puts footnotes back into each individual essay.

## Using a Table of Contents to order the EPUB

In the step above, the names and order of the essays added to the EPUB output is set by using a simple text file (named ESSAYS).

For the INC workflow, we created a custom script (expand-toc.py) that allows a table of contents file written in markdown format to order the contents of the EPUB. The script expects the name of a table of contents file containing (markdown formatted) links to the essays in the order they should appear.

As an additional feature, the table of contents itself is output first allowing a customized table of contents display beyond the automatic table of contents feature of pandoc. To do this, the script remaps the links to the first id found in the linked-to file; for this reason it's important that the individual files (or at least their filtered output, see below) begin with a unique heading.

To customize the output of each file to include alterations developed in the previous step (such as patching the footnotes & using a custom template), the expand\_toc script has a --filter option that should point to a bash script that receives the input filename as the first parameter and which should output to *stdout*.

Finally, expand-toc.py has a --section-pages option that outputs separate pages for each section header in the table of contents.

Open the file essay.sh, it should contain the following:

#!/bin/bash  
  
pandoc \  
 --to markdown \  
 --template essay-template-02.md \  
 --no-wrap \  
 --base-header-level 2 \  
 --id-prefix=$1- \  
 $1

Make sure to make this script is executable:

chmod +x essay.sh

Now you can use the expand\_toc script to create the compiled reader.md document:

scripts/expand\_toc.py toc.md --filter ./essay.sh --section-pages > reader.md

And convert this into an EPUB:

pandoc --self-contained --epub-chapter-level=2 --toc-depth=2 -o reader.epub reader.md

## Installing make

The GNU *make* utility is a program that can help orchestrate your build scripts. See the [related blog post on how make has been used in the INC subgroup](http://digitalpublishingtoolkit.org/2014/10/make-book/).

Mac: One way to install make is to install Apple's XCode development tools. This is either available on your Mac's original system discs, via the Apple App store, or via the Apple developers website. On recent systems, you can simply open the Terminal (in Applications/Utilities) and type:

make

A message should then tell you how to install the program. If make is correctly installed you will see the message:

make: \*\*\* No targets specified and no makefile found. Stop.

Ubuntu:

You may need to install the "build-essentials" package:

sudo apt-get install build-essentials

Debian:

Make is likely already installed, try running it from the commandline.

## Create a makefile

A *Makefile* can be seen as a kind of [executable notebook](http://zgp.org/static/scale12x/) that helps organize ad hoc build scripts into a format that understands how the pieces fit together as *targets* and *dependencies*.

Open the file *makefile*, it contains the following:

sources=$(shell scripts/expand\_toc.py --list toc.md)  
  
all: reader.epub  
  
reader.md: toc.md $(sources)  
 scripts/expand\_toc.py toc.md --filter ./essay.sh --section-pages > reader.md  
  
reader.epub: reader.md styles.css metadata.xml  
 pandoc \  
 --self-contained \  
 --epub-chapter-level=2 \  
 --epub-stylesheet styles.css \  
 --epub-metadata metadata.xml \  
 --toc-depth=2 \  
 -o reader.epub \  
 reader.md

*Devlish detail*: The indented lines of a makefile *must* use the tab character (and not spaces). In SublimeText, you can select text to "show invisibles". You should see the long unbroken dash of a tab character before each command. Be careful when cutting and pasting code into a makefile that no spaces get introduced or make will starting inexplicably complaining.

Note how the two command lines from the build02.sh script have been turned into rules of the form:

target: dependencies ...  
 command(s) to build the target

To use the makefile, simple type:

make

To trigger the build process. By default the all script will build creating first the reader.md, and then reader.epub.

In the first line, the expand\_toc.py script's --list option is used to produce the list of markdown sources in a format (one file per line) that is usable in the makefile. These sources are listed as a dependency to *reader.md*. If you alter one of the linked to markdown sources, *make* will know that it needs to rebuild first *reader.md* and then *reader.epub*. If you subsequently alter just the *styles.css* file, however, only the final step will be repeated to update *reader.epub*.

## Add EPUB trailers as a target to the makefile

A major benefit to using a makefile is the ability to produce a number of different outputs by creating multiple "targets" and rules based on the same sources. In this case we use the [epubtrailer script](http://digitalpublishingtoolkit.org/2014/10/epub-trailers/) to take the epub resulting from pandoc, and create a GIF-format book trailer.

To use the epubtrailer.py script, you will need the Python Image Library, as well as the images2gif python library. If you have installed the python package manager *pip*, you can install both with the command:

sudo pip install PIL images2gif

Now add the following lines to the end of your makefile:

reader-trailer.gif: reader.epub  
 scripts/epubtrailer.py \  
 --width 320 \  
 --height 240 \  
 -o reader-trailer.gif \  
 reader.epub

As mentioned above (see *devlish detail*), make sure that you put an actual single tab character before "scripts/epubtrailer.py" (cutting and pasting from the web version will be using spaces!). Now type:

make reader-trailer.gif