Intro

Good afternoon! My name is Dinah Handel and I'm going to be talking a little bit about using technology- specifically tools like bash, the command line, ffmpeg, and mediainfo, to build simple scripts to process video files. I'm really excited to be here, because like some of you, and all of the panelists, I don't come from a tech background. I studied women's history in undergrad, and then got a library degree. It wasn't really until I started working with audio visual materials as part of my national digital stewardship residency last year that I started to learn how to piece together different tools to get a computer to do work for me so that I didn't have to do it myself, and so that I didn't have to buy expensive software to do it for me either.

This is all to say that I am self taught, and teach myself more every day. I don't work with a/v anymore in my day job, but I still spend a lot of time trying to learn new things, and troubleshoot all the problems I have implementing these new things. I want to underscore that when you are working with technology, in any capacity, you will probably make mistakes, you won't understand what something means at first, and you might spend an hour reading through stack overflow conversations about cron jobs and stop feeling like you've somehow actually lost knowledge instead of gained it, because that is just the process of acquiring a new skill and learning, technology or otherwise. I used to believe the myth that people sat down at their computer, looked at their screen, and wrote perfect code that worked the first time they ran it. I now know this is far from true. Learning to work with technology has been one of the most rewarding aspects of my professional career thus far, and it has helped facilitate my growth into the information professional I am today. I really liked two things from yesterdays women in tech panel, when Lauren Sorensen said something to the effect of "No one's an expert because we're all always learning" and when Mona Jimenez talked about how troubleshooting can be an immensely rewarding, and empowering, feeling.

In the next few minutes, I'm going to talk about a framework that I use to approach creating and writing scripts, and then break down a simple script that I wrote line by line, so that you can see what each component does. Then if there's still time, I can do a short demo of the script so you can see the outcome.

Microservices

A concept that is really fundamental to how I approach using technology and scripting is the idea of microservices. Micro-services break down extensive, multi-step processes inherent in digital preservation workflows into distinct pieces. A simple way to understand microservices is to think about its opposite- the monolith software application. I like this blog post's explanation a lot: compared side by side, "a monolithic app is One Big Program with many responsibilities," whereas "microservice-based apps are composed of several small programs, each with a single responsibility." [1] Micro-services also provide freedom from the constraints of monolith software architecture, which imposes a workflow and process on the user.

Some examples of microservices tasks that might be useful: transcoding a service file from a master file, creating technical metadata, moving metadata and videos into a file together. Once you have these pieces, you can put them together into one script, and then you can theoretically just run that on as many files as you want. These are the tasks that are accomplished by the script that I'll be demoing in a minute.

I think that microservices are really useful when learning how to build scripts, because it allows for you to focus on one task at a time, and then piece those things together as your skillset and needs grow.

Writing a script

So, as mentioned earlier, most of the time, people don't just sit down and write the most perfect script right away. In my experience, and in others too I think, its more like, you write something, it doesn't work, you troubleshoot

it, and then try it again until you get it working, and then you sit down and write more, that doesn't work with what you already have, so then you just keep trying, fixing and making it work as you go.

Something that I like to do when I'm writing out a script is to start by making a list of everything I want the script to do, as detailed as possible. So, for example, something like this:
I want to be able to input one or many files
I want each file to be moved into a directory called videofiles
I want to create an access copy that is also in the directory called videofiles, with the word access appended to its filename
I want there to be another directory that contains technical metadata for both of the files called metadatafiles
I want both of those directories to be in one directory, named after the file name of the file I input
I want checksums to be created for both video files and to be stored in a file called checksum.md5
I want all these packages to be on my desktop

So now that we have our needs for the script, we can start building it. Let's look at the script and I'll break down how we fulfill each task. We're going to be writing in bash, which is a scripting language. You can write scripts in a lot of different languages, this just happens to be the one I know. We're also working with ffmpeg, which is an open source software for working with video files- detecting errors, transcoding derivatives, basically it is very powerful and useful- a great resource for working with it is ffimprovsir (an amia 2015 hack day project), and we'll also be using mediainfo and mediaconch to get technical metadata reports about our files, and finally, md5deep to generate checksums.

Because we are writing the script in bash, we need to have a way to tell the computer that is the language we are using, so we place this message up at the top of our script.

Let's start with the first need- we have to be able to have one or many inputs: we make that possible by starting our script off with a while loop- so up here we are basically saying while the input is not equal to nothing, so when there's something there, do this set of actions, and then we end it at the bottom with a done. When we run the script, it will look for inputs and run the actions on the inputs until there are no inputs left.

Next we're going to set some variables and make the directories that we'll be moving files into. A variable is like a placeholder, so you can set something equal to a particular value, and instead of using that whole something else, you can just use the variable. For example, we are setting the media ID to be the file name of input file without the file extension. We are doing this by typing mediaid in all caps, and then performing a function that takes the basename of the input and removes the file extension. From now on, the media ID will always be this set value. I hope this makes sense.

Next, we're going to set up our directories that we will put files into. We begin by setting up the package directory on the desktop. Then we can use the package variable to set up the rest of the subdirectories.

Next we move onto the transcoding of the access copy using ffmpeg. The ffmpeg command structure is to call the program ffmpeg, and then following the -i is the input (our video file), the flags that specify what we are going to do to the video file, and then the output, which we defined in a variable above the script (remember that you have to define a variable before you try and use it!)

Once we've created our access copy, we'll the make metadata for each of the video files, and distinguish between them through their files names. We'll pipe the output created by mediaconch into an xml formatted document, because later on, the xml file will be much simpler to parse out.

Then, we'll move the video files into the videofiles directory, and move into that directory and create checksums for the video files, and this document will also contain digital forensics information about the computers the files were created on. Finally, we'll alert the user that the script is completed, and the while loop will finish, and if there's another video file to process, it will start over.