# Title Slide

Good afternoon and welcome back to EDirect for PubMed! Today is Part Three: Formatting Results and Unix Tools.

My name is Kate Majewski, and I’m a librarian at the National Library of Medicine in Bethesda, Maryland.

# Remember our theme…

If you remember back to last week, you’ll remember the theme: getting the PubMed data you need, and only the data you need, in the format you need.

# EDirect for PubMed Agenda

Last week, we started the process by getting PubMed data using esearch and efetch. In the next session we’ll restrict our output to only the data we need. Today, we’re going to get our data formatted to our specifications.

This class is all about the details.

# Today’s Agenda

We will begin today with a quick recap of Part Two.

Then, we’ll take a look at how to customize our output format: specifically, customizing separators with –tab and –sep.

We’ll look at how to group related elements together with –block.

And we’ll finish up by talking about some file management techniques, including saving your results to a file and incorporating information from a file into your scripts.

# Recap of Part Two

Last class, we talked about xtract, which lets us pull data from XML and arrange it into a table.

We use the –pattern argument to determine the rows for our table.

We use the –element argument to determine the columns.

# Recap of Part Two (cont’d)

When using these or any of the xtract arguments that specify particular parts of an XML document, we specify an XML element by using its name. Make sure your spelling and capitalization are correct.

We can identify specific elements that are in a particular location in the hierarchy using Parent/Child construction, and we can identify attributes using the @ sign.

# Questions from last class? Homework?

Does anyone have any questions about any of the content we talked about last time, or about the homework?

**[PAUSE FOR QUESTIONS]**

Okay, let’s start looking at how we can customize our output format to make our data pretty!

(SWITCH TO CYGWIN)

Say you want to retrieve a few PubMed records and extract the PMID, ISSN, and the last names of all of the authors for each record.

We start with an efetch. We want one row per PubMed record, so we know what our pattern is going to be. And then we define our elements.

(DEMO IN CYGWIN)

efetch -db pubmed -id 24102982,21171099,17150207 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID ISSN LastName

(EXECUTE)

When we look at the output, though, it’s not as pretty as we’d like. We might expect to have three columns here (PMID, ISSN, LastName), and we do, but it’s hard to tell where each column begins and ends. The columns are all separated by tabs, but so are the multiple values in the LastName column.

We can fix this, using some special formatting arguments.

# -tab/-sep

(SWITCH TO SLIDES)

-tab and –sep let you control what xtract will use as separators.

–tab changes what character is used as the separator after each column

–sep changes the separator between multiple values in the same column

You’ll notice that I phrased those two definitions a little differently: the -tab is the separator **after** each column, the -sep is the separator **between** multiple values. This is going to be important in a little bit, but for now, just remember the headline, which is the -tab separates columns and the -sep separates multiple values in the same column.

By default, both of these separators are a tab character, which is what we’ve seen in all of our examples so far.

# -tab vs. -sep (ANIMATED)

Let’s look at what happens to our output when we change the –tab and –sep argument.

First, we’ll start with no –tab and no –sep.

Here’s our output, and remember that xtract creates a new column for each different element we specify.

**[CLICK]** This might help you visualize it a little better. We have three columns, PMID, ISSN, and LastName, but our last column has multiple values in it.

We can see that there is a tab between each of our columns (PMID, ISSN, LastName). We can also see that there is a tab between each of the values in our LastName column.

# -tab “\t” -sep “\t”

When we add a –tab and a –sep, we don’t see any change. That’s because we defined both our –tab and our –sep with this backslash-t in quotes. That’s shorthand for the tab character, and it’s the default for both of these arguments, which explains why there’s no change.

# -tab “\t” -sep “ ”

Now let’s change the –sep argument from tab to a blank space. We can see that our columns are still separated by tabs, but the multiple values in our LastName column are now separated by single spaces instead.

# -tab “|” -sep “ ”

Let’s leave the –sep the same, but change our –tab argument to the pipe character. Now our columns are separated by pipes instead of tabs!

# -tab “|” -sep “, ”

Using –tab and –sep, we can really customize our output format. My columns are now separated by pipes, but the multiple author last names are separated by comma-space.

# With -tab/-sep, order matters! (ANIMATED)

Customizing these separators can be tricky. I had a lot of trouble figuring out why certain commands were outputting what they were until I realized a couple of things.

First, when you use a -tab or a -sep, it only affects the part of the command that comes after it. **[CLICK]** So, in this example, even though we set our tab to “|” right here, the first two columns are still separated by a tab character, because we didn’t change it until after the first -element.

# With -tab/-sep, order matters!

Second, you can overwrite an earlier -tab or -sep later on in the line. Again, if we look at this example, we start off with the default tab, then change it to | between the second and third columns, and then change it again to “:” between columns three and four. These later -tabs don’t change the output of the first part of the line, but they change the output of everything that comes after them.

 **[PAUSE FOR QUESTIONS]**

# Exercise 1

Please refer to your handout because you’ll want to copy and paste the efetch command.

Write an xtract command that, has a new row for each PubMed record, and has columns for PMID, Journal Title Abbreviation, and Author-supplied Keywords.

Each column should be separated by "|".

Multiple keywords in the last column should be separated with commas.

# Exercise 1 Solution

 (DEMO IN CYGWIN)

efetch -db pubmed -id 26359634,24102982,28194521,27794519 -format xml | \

xtract -pattern PubmedArticle -tab "|" -sep "," -element MedlineCitation/PMID ISOAbbreviation Keyword

(EXECUTE)

If you’re still working on Exercise 1, I’m going to ask you to pause for now. Remember that the answers to all of the exercises are at the bottom of your handout, so you can go over them later if you want, and we’ll make sure to get the recording up as quickly as possible, too.

# Getting Author information

(STAY IN CYGWIN)

Let’s say we want to pull all of the authors for each citation in our results set. Again, we want one row per record, and we’ll put the PMID in there, too

We could try this command:

(DEMO IN CYGWIN)

xtract –pattern PubmedArticle –element MedlineCitation/PMID LastName Initials

(EXECUTE)

But this doesn’t work the way we want. It gives us all of the author last names for a record, then all of the initials. What we want is to retain the relationship between an individual last name and the corresponding initials.

# xtract-ing authors (ANIMATED)

Let me show you visually what’s happening with that code. On the left, we have some dummy XML. On the right is going to be the output of our command. On the bottom is our code.

**[CLICK]** First, xtract finds the first instance in the XML of our –pattern element. In this case, that’s PubmedArticle. Then, xtract looks for all instances of the element or elements identified in the –element argument, and outputs them.

**[CLICK]** First, it will look for MedlineCitation/PMID, of which there should only be a single occurrence.

**[CLICK]** Then, it will look for LastName. It finds one here, “Wu”, and outputs that. But then, rather than giving us the Initials, **[CLICK]** it gives us the next LastName.

**[CLICK]** **[CLICK]** It keeps on going through all of the LastName elements in the pattern until it can’t find another one. **[CLICK]** Only then does it look for Initials.

**[CLICK]** **[CLICK]** **[CLICK]** Again, it gives us each Initials element in the pattern until it can’t find another one.

# -block

To fix this, we can use the –block argument.

–block is one of a series of xtract arguments known as exploration arguments, which means it will help us identify and group the elements we want to output. This is exactly what we want: to group each related pair of LastName and Initials.

–block associates multiple child elements of the same parent element in the results.

(SWITCH TO CYGWIN)

(DEMO IN CYGWIN)

xtract -pattern PubmedArticle -element MedlineCitation/PMID -block Author -element LastName Initials

(EXECUTE)

(SWITCH TO SLIDES)

Here’s how –block works:

**[CLICK]** Just like before, xtract finds the first instance in the XML of our pattern, PubmedArticle.

**[CLICK]** Then, xtract looks for all instances of the element or elements identified in the –element argument, and outputs them. This time, though, that’s just MedlineCitation/PMID.

**[CLICK]** Then, xtract finds the first instance of the element identified in the –block argument, which is Author.

**[CLICK]** xtract looks within that first Author element, and outputs the element or elements identified in the –element argument. In each author, there should be one LastName element, **[CLICK]** and one Initials element.

**[CLICK]** Then, xtract looks for the next Author. **[CLICK]** xtract looks within the second Author element, and outputs the Last Name and **[CLICK]** Initials.

**[CLICK]** **[CLICK]** **[CLICK]** **[CLICK]** **[CLICK]** **[CLICK]** xtract keeps repeating these steps until there are no more Author elements in the pattern

If there isn’t another Author element in the pattern, xtract moves on. It has reached the end of the line, so it goes back to the beginning, and finds the next instance of the element identified in the –pattern. And so on.

**[PAUSE FOR QUESTIONS]**

# This is good, but we can do better

So, this isn’t a bad way to add author information to a table with xtract, but when I need to do this, I do it a little bit differently.

This is definitely a pretty good first draft: We’ve got all of the authors for each record on a single line, and we’ve got each author’s last name and initials grouped together…well, sort of.

We have last name/initials, last name/initials, but they’re not really grouped. Everything’s separated by tabs. Given what we know about -tab and -sep, we can do a little better. We just need to learn one new trick.

# What we know so far… (ANIMATED)

From the beginning of this class, we’ve talked about using –pattern to create rows and –element to create columns.

So far, each of our columns have had data from a single element or attribute in them. If you look at the three columns of this output table, **[CLICK]** you can see that happening. The first column only has data from the PMID element. The second column only has data from the ISSN element. The third column has multiple values, but each of those values is from different occurrences of the same element, the LastName element.

However, with what we know about –tab and –sep, we can now specify different characters to separate BETWEEN columns vs. between multiple values in the SAME column.

So what if we put values from two DIFFERENT elements in the SAME column? We could separate them with a custom delimiter using -sep, because remember, -sep separates multiple values in the same column.

This way we could actually “group” each set of last name and initials in their own column.

# Putting two different elements in the same column

We can do this with a comma. Instead of separating the multiple elements with a space, we use a comma instead, which puts them both in the same column. We’re telling xtract to not consider these separate columns, and to use the –sep character to separate last name and initials.

When we combine this with –block, we still have our columns separated out by tabs, but our last name and initials are grouped together in the same column, with a space separating the two values, which makes our author information a lot easier to read!

How -block creates columns

Now the reason this works is because of how xtract puts blocks into columns. We said that the comma groups last name and initials into the same column.

Based on what we know, then it seems like we should only have two columns here, one for the PMID from this -element argument, and one column created by this second -element argument.

But remember what I said before: -tab defines the separator **after** each column, after each -element. In this case, when you use the -block argument, that means that, for each block xtract goes through (so, for each Author), xtract looks for all of the LastName and Initials elements inside that block.

When it can’t find any more, it’s done with that -element argument, and puts that -tab character at the end to separate it from the next column. This means that, when we use the -block argument, whenever we get to the end of a block in our output, we’re going to create a new column.

However, when we get to the end of the -pattern, xtract knows not to print that extra -tab at the end, and instead replaces it with a line break.

# “-block” resets -tab/-sep to default (ANIMATED)

There’s one other thing to know about using -block with -tab and -sep that can also change your separators, and if you don’t pay attention to it, you might wonder why things aren’t working the way you expect.

Whenever you have a “-block” argument in your command, it resets your -tab and -sep to the default, which is the tab character.

So looking at this example, **[CLICK]** we define our -tab as “|”, and our first two columns are separated by the pipe.

However, once we start creating new columns for each of these Author blocks, our -tab has been reset to its default. This is because the -block argument has reset it. Our -sep still works, because we defined it after our -block.

**[CLICK]** If we want all of our columns separated by pipes, we need to go in and add another -tab argument after our -block argument.

**[PAUSE FOR QUESTIONS]**

# Exercise 2 (5:00)

Enough show-and-tell. Time for another hands-on exercise. This one’s a little tricky, as we’re going to combine –tab, -sep, and –block all together.

Write an xtract command that has a new row for each PubMed record, has a column for PMID, and lists all of the MeSH headings for each record, separated by “|”. If a heading has subheadings, separate the heading and each of the subheadings with a “/”.

# Exercise 2 Solution

(DEMO IN CYGWIN)

efetch -db pubmed -id 24102982,21171099,17150207 -format xml | \

xtract -pattern PubmedArticle -tab "|" -element MedlineCitation/PMID

[WHEN DONE] If you figured out the block on that one, you are doing great. If you figured out block AND the comma, you are doing really great. If you figured out the block, the comma AND the tabs and seps, you are an impressive human being or you have done this before.

If you didn’t figure out any of that on your own but you looked at the answer and sorta kinda get the idea of what we’re doing, that’s about what I’d hope for at this point.

This takes practice. If you’re still having trouble, take a breath and make peace with the idea that you will need to review, practice, and if you’re like me, sleep on it and let your brain work through it.

For now I’m going to change gears a little bit, and talk about Unix: Unix tips that will be useful in using EDirect.

You may notice, depending on the size of your terminal and your screen, that we have some unfortunate line wrapping, and otherwise somewhat unwieldy results. Look at your last results - And that’s only with three records!

# There is only so much data manipulation we want to do here in EDirect. One of our goals here is to extract our data as neatly, cleanly, and as well organized as possible to import into Excel or a text editor or some other familiar environment where we can “play.”

# Copying and pasting is not a very practical option for many of the projects you may be brewing.

# Fortunately, saving our output to a file is simple.

# Saving Results to a File

(STAY IN CYGWIN)

We just take that same set of commands we had before, and add the greater-than symbol, a space, then a file name. This will save your file in the current directory.

(DEMO IN CYGWIN)

efetch -db pubmed -id 24102982,21171099,17150207 -format xml | \

xtract -pattern PubmedArticle -tab "|" -element MedlineCitation/PMID > filename.txt

(EXECUTE)

Remember, Unix isn’t always very forthcoming with what it’s doing. When we execute this, it looks like nothing happened, but it did actually save the file to my current directory.

By way of proof, we can use another Unix command: ls. “ls” lists all the files in your current directory.

(DEMO IN CYGWIN)

ls

(EXECUTE)

When we do that, we can see the file that we just made is right there! And I can also just open this file and we can see the what’s in there.

(SHOW FILE IN TEXT EDITOR)

You can save in whatever file format you want. Say I wanted to do an esearch, then use efetch to get a list of PMIDs. If that list was really long, I could use the greater-than sign to redirect the output to a text file, like this.

(DEMO IN CYGWIN)

efetch -db pubmed -id 24102982,21171099,17150207 -format uid > testfile.txt

(EXECUTE)

If I was using efetch to download full XML, I could save the output to an XML file, like so.

(DEMO IN CYGWIN)

efetch -db pubmed -id 24102982,21171099,17150207 -format xml > testfile.xml

(EXECUTE)

File extensions are really useful for some operating systems, but Unix doesn’t generally require them, so I could even save this with no extension at all!

# But where is my file!?

(STAY IN CYGWIN)

If today is the first time you’ve opened up the Unix terminal on your machine, and you’ve never used Unix before, you may be a little lost as to where your files are. Don’t worry; I had this problem too!

Without going into some of the more advanced file management techniques that Unix has to offer, let me provide a few suggestions on a way to find your file.

You can try the command “pwd”, which “Prints the Working Directory”: it displays on screen the name of the directory you’re currently working in. That might give you some clues as to where your file is.

For Cygwin users, you can try running the command:

cygpath -w ~

This command tells me where in Windows my Cygwin directory lives.

For Mac users, your working directory is probably in the Users folder (e.g. Users/<your user name>).

# Another way to find your files

Alternatively, you could try saving a file to that location with a particularly distinctive name, and then search your computer for it using your operating system’s native search feature.

Let’s all walk through this method together.

(SWITCH TO CYGWIN)

It’s important to figure out where this directory is, because in just a moment we’re going to try reading from files stored in this directory, so we need to know where we are.

Also, we’ll be doing another demo in a few minutes where we try to pull information from a file to include in one of our commands, so this will help us all have the same file to work with.

Run the following command.

(DEMO IN CYGWIN:)

efetch –db pubmed –id 24102982,21171099,25359968,17150207 \

–format uid > specialname.csv

(EXECUTE)

That should create a .csv file in your working directory named specialname.csv

Again, we can verify that the file is there by using the unix command “ls”.

(DEMO IN CYGWIN: ls)

(EXECUTE)

And there we can see our file.

**[PAUSE FOR QUESTIONS]**

# Exercise 3: Retrieving XML

(SWITCH TO SLIDES)

How can I get the full XML of all articles about the relationship of Zika Virus to microcephaly in Brazil? And please save your results to a file.

# Exercise 3 Solution (ANIMATED)

And here’s our answer. **[CLICK]**

If you’re still having trouble with finding your file, or with that last exercise, we can answer some of those questions at the end of the session, but we’re going to move on now, so I’ll ask you to stop working on it for now.

Remember how we said that Unix was really good at working with files, and has lots of tools designed just for manipulating files, but that we weren’t really going to talk about a lot of them? The command “cat” is an exception.

# cat

cat is short for concatenate. It can be used to open files and display them on screen, and it can also combine or append files together.

(DEMO)

Before the class, I created two files, named file1.txt and file2.txt.

(OPEN file1.txt, file2.txt)

You can see that file2.txt is just the word “hello”, and that file2.txt is just the world “world”.

(SWITCH TO CYGWIN)

We can use cat to see the contents of file1.txt by simply typing “cat file1.txt” and executing.)

(DEMO IN CYGWIN: cat file1.txt)(EXECUTE COMMAND)

We can do the same with file2.txt.

(DEMO IN CYGWIN: cat file1.txt)(EXECUTE COMMAND)

We can also use “cat” append one file to the other:

(DEMO IN CYGWIN: cat file1.txt file2.txt)(EXECUTE COMMAND)

And you see our output is the contents of file1.txt followed by the contents of file2.txt.

We could then take that same cat command, and use the greater-than symbol to redirect the output of the command to a third file.

(DEMO IN CYGWIN: cat file1.txt file2.txt > file3.txt)(EXECUTE COMMAND)

And if open that new file up:

(DEMO IN CYGWIN: cat file3.txt)(EXECUTE COMMAND)

We can see that it contains the contents of both of the other files.

So how does that help us?

# Reading a search string from a file

(STAY IN CYGWIN)

For this next demo, I’m going to ask you **not** to follow along with me. Just watch for a moment.

Say you have a long search string that you want to use in esearch. For example, I have a text file named “searchstring.txt”

(OPEN searchstring.txt)

This file contains a search string that is an effort to capture all intramural research coming out of one of the other National Institutes of Health, the National Institute of Allergy and Infectious Diseases, a.k.a NIAID. You may have seen them in the news discussing Zika Virus, Ebola, or other nasty diseases.

NIAID has lots of different names, labs, subcomponents, and divisions, and any of them could be listed in a paper’s affiliations. Additionally, there are over 100 researchers who work at NIAID.

The search strategy might change over time (you might need to add new authors or new names for subcomponents), so you don’t want to have this long, complex search string be a part of your actual esearch command. It would be unwieldy and difficult to edit.

However, you can use “cat” to pull the contents of this file into your search string.

Again, since you don’t have this searchstring.txt file on your computer, I’m going to ask you to just watch for now and not follow along, since this isn’t going to work for you without an example file.

You would start your esearch normally, up to the “-query.”

(DEMO IN CYGWIN: esearch –db pubmed –query)

Now, we want to include the contents of searchstring.txt.

(DEMO IN CYGWIN: esearch –db pubmed –query “searchstring.txt”)

(DO NOT EXECUTE)

Now this isn’t right, because this will only search for the string “searchstring.txt” in PubMed. We need to use “cat” to read the contents of the file, not just search for the name of the file.

(DEMO IN CYGWIN: esearch –db pubmed –query “cat searchstring.txt”)

(DO NOT EXECUTE)

This isn’t much better, since it just searches for the string “cat searchstring.txt” What we really want to do is use the output of the command “cat searchstring.txt” as our search string.

(DEMO IN CYGWIN: esearch –db pubmed –query “$(cat searchstring.txt)”)

The dollar sign/parentheses gives us the value of what’s inside the parentheses. The value of “cat searchstring.txt” is the output of reading searchstring.txt to the screen.

(EXECUTE)

# Reading a list of PMIDs from a file

(STAY IN CYGWIN)

Now, you could do this in efetch, too: Let’s say you had a list of PMIDs in a file, and you wanted to fetch them all. You might want to do this if you maintain a list of citations outside of PubMed that you want to retrieve, instead of just retrieving search results.

If the list in your file was a long string of PMIDs, separated by commas, you could do the exact same thing. But let’s say you have a file (like a .csv file), with one PMID on each line. If you do this the same way, you’ll get an error.

For this, we want to use another EDirect command we haven’t yet talked about, but before I do that, we need to delve a little bit deeper into something we’ve already talked about a lot, which is the pipe.

# EDirect and the History Server (ANIMATED)

(SWITCH TO SLIDES)

When we introduced the pipe in the first class, we were talking about esearch and efetch, and we said we were just piping the results of one command and using them as the basis for another. If you want to think of it like that, that’s totally fine.

But remember, when we do an esearch, we aren’t actually getting out a list of PMIDs. **[CLICK]** We got an XML snippet with the number of PMIDs that met our search criteria, plus a WebEnv/QueryKey.

What is happening is that the results of the esearch, including the full list of PMIDs is being stored to the history server. The History server is what keeps track of your previous searches, just like in Advanced Search Builder in the web version of PubMed. The WebEnv tracks your session (to identify your specific history, and not some other user’s history). The QueryKey indicates which query in your history this is. **[CLICK]**

What the pipe is actually doing **[CLICK]** is piping the WebEnv and Query Key from esearch to efetch. The esearch is saving the results on the History Server **[CLICK]** (including both the PMIDs and the fact that they come from PubMed), and the WebEnv and Query Key tell efetch which results to pull down from the history server. **[CLICK]**

Again, most of the time you can just think of the “|” as piping PMIDs from one command to the other, but the fact that this works this way actually allows you some more interesting and powerful options, because esearch isn’t the only thing that can put PMIDs on the history server.

**[CLICK]** The EDirect command “epost” can be used to upload a series of PMIDs to the History server.

Piping the results of epost to efetch works exactly the same way as esearch

# epost

(SWITCH TO CYGWIN)

(DEMO IN CYGWIN: epost –db pubmed –id 24102982,21171099)

So here’s our epost command, with a few PMIDs specified. Make sure we specify the database, using –db pubmed: we need to tell epost that these are PMIDs, not UIDs from a different database.

(EXECUTE)

If we just run epost by itself, we get a WebEnv and Query Key.

# An epost-efetch pipeline

(STAY IN CYGWIN)

We can then pipe the results of that into efetch, just like we would with an esearch:

(DEMO IN CYGWIN: epost –db pubmed –id 24102982,21171099 | efetch –format abstract)

Of course, that doesn’t get us anything that efetch doesn’t.

But when we use cat to open a file, then pipe the contents of the file into epost, then pipe the contents of that into efetch, we’re in business:

(DEMO IN CYGWIN: cat specialname.csv | epost –db pubmed | efetch –format abstract)

This uses cat to get the PMIDs out of the file, pipes the PMIDs to epost, which uploads them to the History server, and the pipes the WebEnv and Query Key to efetch, which uses it to retrieve the correct PMIDs from the History server and download the records.

# Using the –input argument

Until recently, this was the best way to get a list of PMIDs into an epost-efetch pipeline. However, NCBI recently added new functionality to EDirect.

Last year, NCBI added a new argument to epost. The –input argument lets you skip the “cat” part. You can specify a file as an –input to your epost, then pipe that to your –efetch!

(DEMO IN CYGWIN: epost -db pubmed -input specialname.csv | efetch -format abstract)

It’s a little shorter, one less step to worry about, and gets you the same results.

(EXECUTE)

# Coming next time…

Next time we’re going to continue our discussion of xtract, and talk about some more advanced arguments that help us limit our output to the exact data we need.

We’re also going cover some ways to put all of what we’ve learned together, and some tips for developing practical solutions to real-world problems.

# In the meantime…

As always, we encourage you to visit our website for more information and documentation on EDirect, as well as the script, slides, and recording, as soon as we can put it up.

Remember to sign up for the utilities-announce mailing list, which you can access from the Contact Us button on our website.

And from that same button, you can send us any questions you have during the week.

# Questions?

But with that, I would be happy to take any questions you have now.