2.13.3 Process Text Streams

Click one of the buttons to take you to that part of the video.

Process Text Streams 0:00-0:09

In this demonstration, I'm going to show you several command line tools that you can use to process text streams.

---

sed Command 0:10-1:56

Let's begin by looking at sed. sed is a text stream editor. Using sed, you can capture a text stream and then modify it. For example, you can search through the text stream for certain words and delete them, or you could look for certain words and replace them with other words.

Let's take a look at an example. We have a file in my /home directory here called customers. You can see that it contains the customer first name, last name, and their phone number. Let's suppose for some weird reason that we need to replace all last names of Jones in this file with Smith.

To do this, we can use sed. First we'll use the 'cat' command to send the contents of the 'customers' file to the input of the 'sed' command. Now we need to tell sed what it is we want to do. Because we want to replace text, we're going to use the 's' option. If you wanted to delete text, you would use d instead of s.

With the s option, first we have to specify what text it is that we want to locate and replace. Remember, we said we wanted to change all instances of the text 'Jones' to 'Smith'. Press Enter, and you can see that one instance was found right here. Mike Jones has now had his name changed to Mike Smith.

It's important to note that we didn't actually change the customers file itself. Let me run a 'cat' on customers again, and you can see that the original file is just fine. What we basically did is created a text stream here with the cat command.

We sent customers to the input of sed. sed made the change to the text stream, not the original file, and wrote the output on the screen. We could have taken and output the standard out from sed to a file, we could have piped it to another command, whatever it is we needed to do.

---

awk Command 1:57-7:36

The next utility we're going to look at is the awk command. awk is really an enhanced version of sed. It's really a lot more powerful as to what it can do. If you need to just do a simple search and replace or a simple deletion, then sed is probably the easier of the two utilities to use. But if you want to perform more complex stream editing, then awk can do things that sed could only dream of.

Before we start looking at awk, you need to understand that awk treats each line in the file as if it were a separate record. Really, what awk does is look at this entire file and treat it as a database. Within that database we have record 1, record 2, record 3, record 4, and record 5.

Remember, within a database, a record is made up of individual fields. awk defines fields based on spaces. The spaces within each line are used as delimiters. We have a delimiter here, and we have a delimiter here. That means we have field 1 here--as far as awk is concerned--field 2 here, and field three here within each record.

The important thing to remember when working with awk is the fact that each of these fields is referenced with a different number. Field 1 is $1, field 2 is $2, field 3 is $3. If we had more text on each line, we would have $4; $5, and so on. One for each field within a given record.

I know this probably sounds confusing, so let's take a look at a simple example of using awk, and hopefully that will make it more clear. The first thing we need to do, as we did with sed, is use the 'cat' command to create a text stream, and we will pipe the output to the awk command.

First, let's simply just use awk to print out only field 2 from each record in the file. To do this, we enter 'awk', and then in tick marks and within curly braces we enter '{print' to specify that we print the output to the standard out of the command.

Then we have to tell it, "What? What do you want to print?" Well, we wanted to print the second field within each record, so we enter '$2}' and closing tick mark.

You can see in this example we took the output of the cat command, sent it to the input of the awk command and then awk said, "Okay, based upon the command I've been given, I'm going to ignore field 1 in each record and field 3 of each record, and all I'm going to do is print field number 2," and that's what it did. Jones, Doe, Johnson, Simmons and Saez.

In addition to just selecting text, awk can also add characters to the text stream and reformat the output in a wide variety of different ways. For example, let's bring up the command that we just ran again. This time we're not going to simply print fields from each record; we're going to insert text into the output and reformat the data.

In this example, we're still going to run cat and send the output of cat to the input of the awk command. We're still going to use print, but now we're going to change things quite a bit.

First of all, we're going to add some literal text to the output. I have to use quotation marks to format it. Everything has to be within closing and opening quotation marks. What I want to do is label each of these fields within each record. We have a first name, a last name, and a phone number.

Let's enter 'First:' that literal text we printed on the screen. Then we need to pull field one from each record. Within quotation marks we enter '$1' to reference field one. Then we also want to insert a 'tab' character.

Notice over here that some names are longer than others. First name here's quite short. This first name is quite long. As a result, the last names don't line up and the phone numbers don't line up. By adding tab characters in, we can make things line up a lot nicer.

This part of the command prints first on the screen and then grabs the first field from each record, prints it, and then adds a tab character (\t). Now we want to do the same thing with the second field. Let's put everything in quotation marks, remember? We type 'Last:'. Then we want to reference the second field within each records. Within quotation marks we put '$2'. We need another tab character '(\t)' again to format things nicely.

Then let's do the same thing with the phone number. Quotation marks, 'Phone:', and we want to reference field 3 this time. Quotation marks, '$3', and we'll add a tab character '(\t)' at the end of the output. Press Enter.

Notice what awk did as a result. For each record, it first printed the text, First:. Grabbed field number 1, Mike in this case. Then it printed a tab, and then it added Last:, and then whatever the value field 2 is, in this case Jones. And then a tab and then Phone:. Then the value of field 3 for each record, and then another tab at the end. This is really just a very simple example of how you can use awk to manipulate the text in a text stream.

You can actually use awk to change the text you're receiving and manipulate it and output it in many different ways--whatever way is useful for what you're trying to accomplish.

The next utility we want to look at is the split command, which basically takes a single file and splits it into a series of files.

---

split Command 7:31-9:46

By default, the split command will split the input find into 1,000-line segments. However, you can also configure it to use a specific number of lines, either more or less than 1,000--whatever it is you need to do.

For example, once again we have our customers file right here. We have one customer on each line. Let's suppose that we want to break this file up into one separate file for each customer. To do this, we run 'split', and then we put in a '-' and specify the number of lines we want to break on. Again, in this case, we want a separate file for each customer, so we're going to break on each line.

I specify a value of '1' here to break every one line. Then we have to specify an output file name. Let's use 'customerfile_', and hit Enter. Oops. You do actually have to specify the name of the file that you want to split. I failed to do that.

We want to split 'customer' file. Each individual file that we split out from customer is going to be named customerfile_, and then the rest of the name will be automatically generated by split. I'm going to press Enter. It also helps if you spell the file name right. It's not customer; it's 'customers'.

Let's try that one more time. That looks a little better. Now let's do the 'ls' command, and we see five new files created here--one for each line in the original customers file. We can use the 'cat' command.

Notice for each file, the ending of the filename was automatically generated. It started with aa and then it went to ab, ac, ad, ae, and so on. We use the 'cat' command to view the first one, and we see that it contains just the Mike Jones record. We can do the same thing for the second file and see the Jenny Doe's record, and so on, all the way down to ae, which contains the record for Lisa Saez.

---

tr Command 9:47-10:39

The last command we want to look at in this demonstration is the tr command. With tr, you can do a lot of different things. It's used to translate or delete characters. For example, you could cause all lowercase characters in the text stream to be translated into uppercase characters.

Let's look at an example of doing this. Let's use the 'cat' command. What we want to do is send the output of the cat command viewing the customers file to the input of the 'tr' command.

In this situation, let's tell it to translate all the lowercase characters a-z to upper case characters A-Z. Press Enter. Notice in the output of the command that all of the lowercase characters in each name in the file were translated from lowercase to uppercase characters.

Summary 10:40-10:51

That's it for this demonstration. In this demo, we talked about processing text streams. We looked at several command line tools you can use to process text streams. We looked at the sed command, we looked at the awk command, we looked at the split command, and we ended this demonstration by looking at the tr command.
