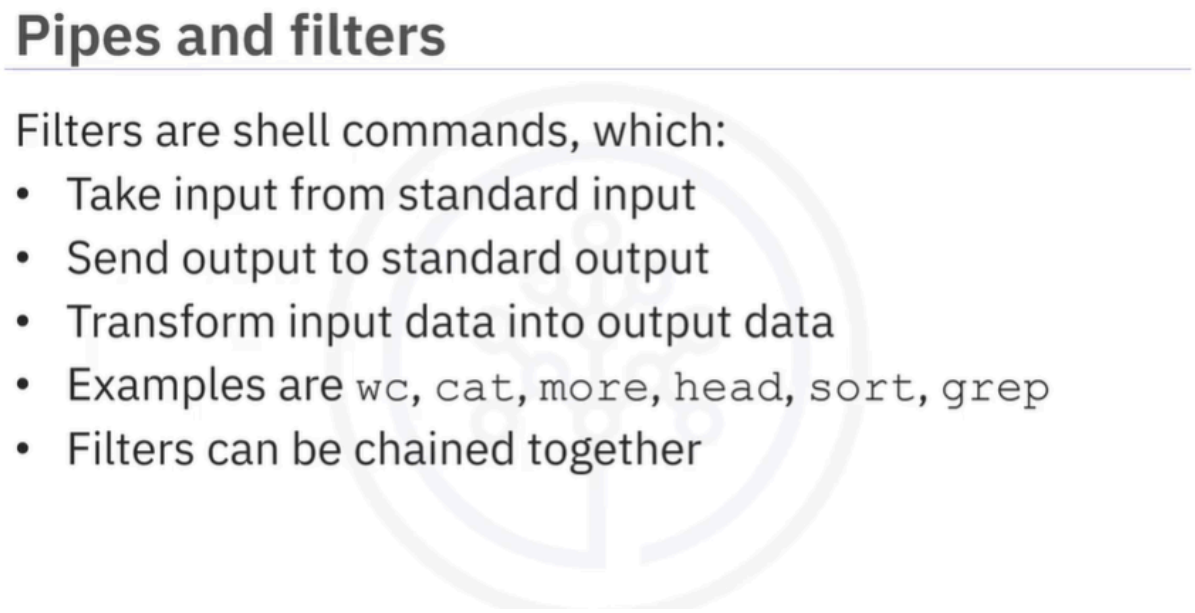
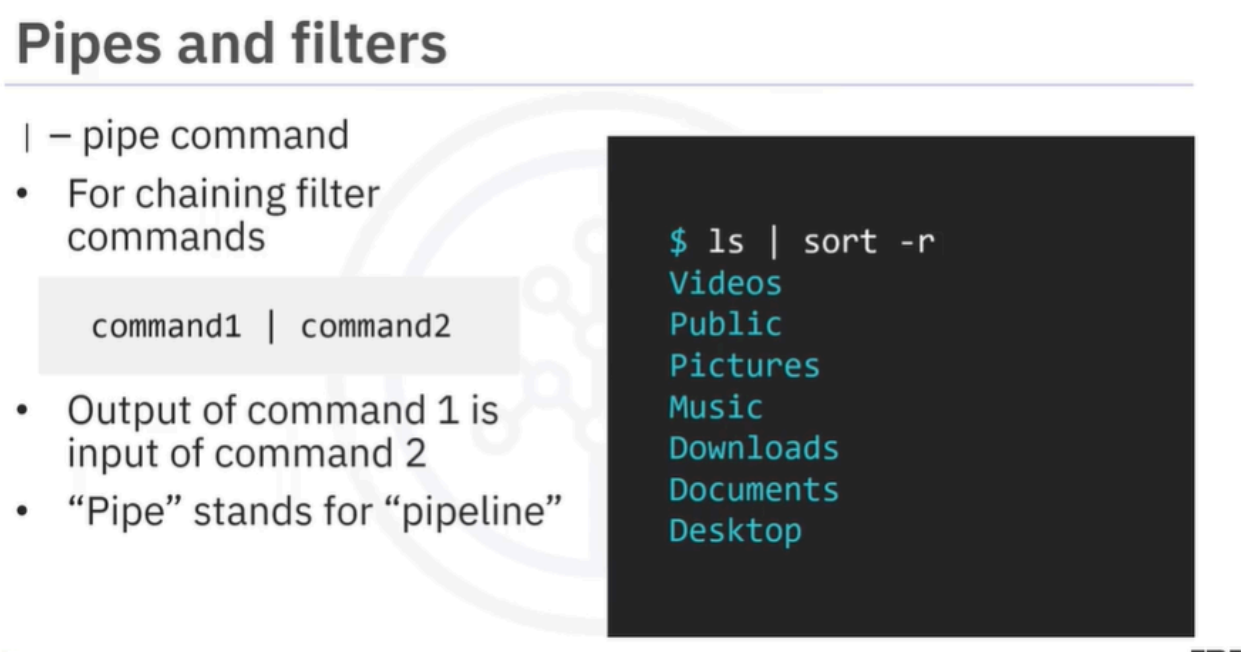
Filters, Pipes, and Variables

Sure! Let's talk about **pipes and filters** in the context of shell commands.

Pipes and filters are like a team of workers in a factory. Each worker (filter) takes in raw materials (input data) and processes them to create a finished product (output data). For example, imagine you have a worker who sorts fruits. You can send a basket of mixed fruits to this worker, and they will sort them into apples, bananas, and oranges. In the shell, commands like `sort`, `grep`, and `wc` are filters that transform data in various ways.

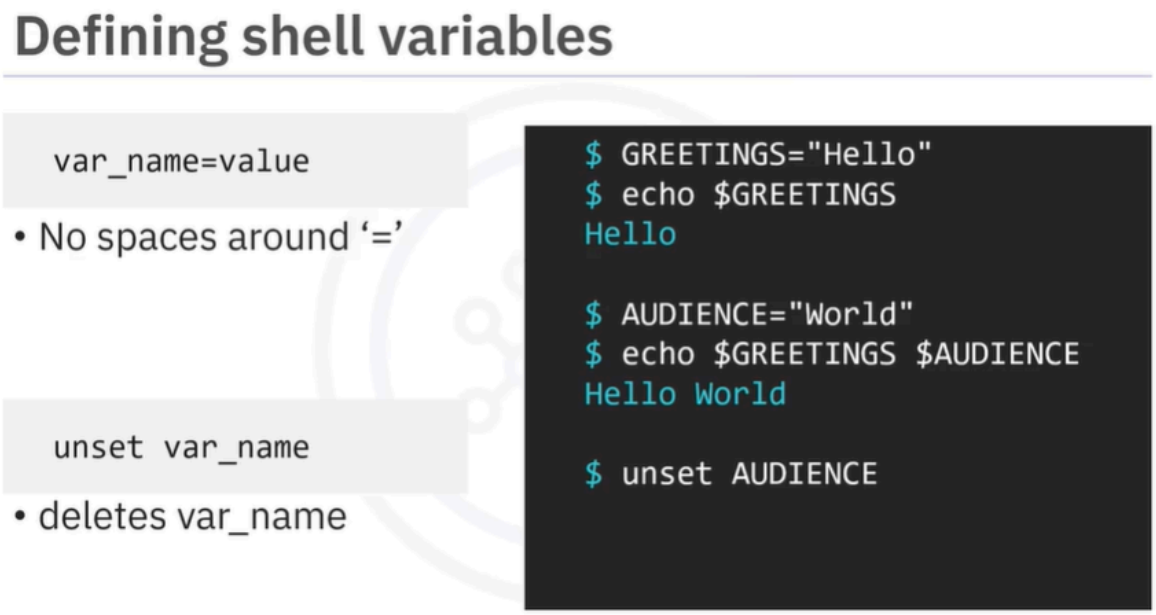
Now, the **pipe** is like a conveyor belt that connects these workers. When you use a pipe (denoted by `|`), it allows the output of one worker (command) to become the input for the next worker. For instance, if you want to list files in a directory and then sort them, you can use the command `ls | sort`. Here, `ls` lists the files, and the pipe sends that list directly to the `sort` command, which organizes them for you.





- Filters or shell commands or programs which take their input from standard input normally the keyboard and return their output to standard output, which is normally the terminal. We can think of a filter as a transformer, a program that transforms input data into output data. There are many examples, including wc, cat, more head, sort, grep and so on. The value of filters is that they can be chained together which brings us to the pipe command.
- The pipe command, denoted by a vertical slash, immensely extends the functionality of shells. It allows you to chain together sequences of filter commands. The use pattern looks like this accordingly, the output of command 1 becomes the input of command 2, and so on. Not surprisingly, pipe is short for pipeline. For example, you can pipe the output of ls to the sort command with the -r option, which results in a reverse sorted list of directory contents

- Shell variables are variables that are limited in scope to the shell in which they were created. Accordingly, shells cannot see each other's shell variables. You can invoke the set command to list all variables and their definitions that are visible to the current shell. Because it also lists a lot of subsequent information, you can pipe the output to head in order to show just the first four variable definitions.



- To define a new shell variable, simply use the equal sign to assign a value to your chosen variable name. Notice that there are no spaces around the equal sign. As an example, let's define a shell variable called Greetings, which stores the string hello. To see the contents of the new variable Greetings, use the dollar sign to access its value, then echo it back. You can display multiple variables as well. Let's define another variable audience with the value world, then echoing both variables back returns Hello World. To clear a variable, use the unset command. For example, unset AUDIENCE deletes the variable AUDIENCE.

- Environmental variables are just like shell variables, except they have extended scope. They persist in any child processes spawned by the shell in which they originate. You can extend any shell variable to an environment variable by applying the export command to it. For example, export GREETINGS makes Greetings an environment variable. To list all environment variables, use the env command. Let's check whether Greetings was exported as an environment variable by piping the output of env to grep and filtering the results using the pattern GREE. Indeed, Greetings is now an environment variable.