Pipes

A picture containing text, clipart

Description automatically generated

What are pipes?

Put simply, pipes are commands in Linux which allow you to use the output of one command as the input of another.

Pipes "|" use the following format:

[command 1] | [command 2] | [command 3] ... | [command n]

(no limit to the number of times in a row u can pipe!)

Pipe examples

Ex 1:

Let's start with a commonly used example.

Recall the commands:

* [sort](https://man7.org/linux/man-pages/man1/sort.1.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMLX0117ENSkillsNetwork860-2022-01-01): sorts lines in input
* [uniq](https://man7.org/linux/man-pages/man1/uniq.1.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMLX0117ENSkillsNetwork860-2022-01-01): prints input with consecutive repeated lines collapsed to a single, unique line

With the help of the pipe operator, you can combine these commands to print all the unique lines in a file!

Suppose you have the file pets.txt with the following contents:

$ cat pets.txt

goldfish

dog

cat

parrot

dog

goldfish

goldfish

If you *only* use sort on pets.txt, you get:

$ sort pets.txt

cat

dog

dog

goldfish

goldfish

goldfish

parrot

And if you *only* use uniq, you get:

$ uniq pets.txt

goldfish

dog

cat

parrot

dog

goldfish

But by combining the two commands in the correct order, you get back:

$ sort pets.txt | uniq

cat

dog

goldfish

parrot

which are the sorted, unique lines from pets.txt!

Ex 2:

Some commands, such as tr, *only* accept "standard input" as input (not strings or filenames):

* [tr](https://man7.org/linux/man-pages/man1/tr.1p.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMLX0117ENSkillsNetwork860-2022-01-01) (translate) - replaces characters in input text.
  + Syntax: tr [OPTIONS] [target characters] [replacement characters]

In cases like this, we can use piping to apply the command to strings and file contents.

With strings, you could, for example, use echo in combination with tr to replace all vowels in a string with underscores, as follows:

$ echo "Linux and shell scripting are awesome\!" | tr "aeiou" "\_"

L\_n\_x \_nd sh\_ll scr\_pt\_ng \_r\_ \_w\_s\_m\_!

To perform the complement of the operation from the previous example, that is, to replace all consonants with an underscore, you can use the -c option like this:

$ echo "Linux and shell scripting are awesome\!" | tr -c "aeiou" "\_"

\_i\_u\_\_a\_\_\_\_\_e\_\_\_\_\_\_i\_\_i\_\_\_a\_e\_a\_e\_o\_e\_

With files, you could use cat in combination with tr to change all of the text to upper case as follows:

$ cat pets.txt | tr "[a-z]" "[A-Z]"

GOLDFISH

DOG

CAT

PARROT

DOG

GOLDFISH

GOLDFISH

The possibilities are endless! For example:

$ sort pets.txt | uniq | tr "[a-z]" "[A-Z]"

CAT

DOG

GOLDFISH

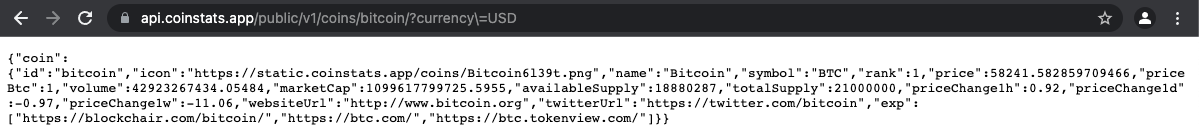
PARROT

Ex 3:

You can even use curl in combination with the grep command to extract components of URL data by piping the output of curl to grep.  
Let's see how you can use this pattern to get the current price of BTC (Bitcoin) in USD.

First, you find a public URL API. In this example, you will use one provided by [CoinStats](https://coinstats.app/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMLX0117ENSkillsNetwork860-2022-01-01" \t "_blank).

Specifically, they provide a public API (no key required) https://api.coinstats.app/public/v1/coins/bitcoin\?currency\=USD which returns some json about the current BTC price in USD.

You can see what this looks like in your browser: 

Entering the following command returns the BTC price data, displayed as a json object:

$ curl -s --location --request GET https://api.coinstats.app/public/v1/coins/bitcoin\?currency\=USD

{

"coin": {

"id": "bitcoin",

"icon": "https://static.coinstats.app/coins/Bitcoin6l39t.png",

"name": "Bitcoin",

"symbol": "BTC",

"rank": 1,

"price": 57907.78008618953,

"priceBtc": 1,

"volume": 48430621052.9856,

"marketCap": 1093175428640.1146,

"availableSupply": 18877868,

"totalSupply": 21000000,

"priceChange1h": -0.19,

"priceChange1d": -0.4,

"priceChange1w": -9.36,

"websiteUrl": "http://www.bitcoin.org",

"twitterUrl": "https://twitter.com/bitcoin",

"exp": [

"https://blockchair.com/bitcoin/",

"https://btc.com/",

"https://btc.tokenview.com/"

]

}

}

(The output is formatted for your convenience in this lab).

The json field you want to grab here is "price": [numbers].[numbers]". To grab this you can use the following grep command to extract it from the json text:

grep -oE "\"price\"\s\*:\s\*[0-9]\*?\.[0-9]\*"

Let's break down the details of this statement:

* -o tells grep to *only* return the matching portion
* -E tells grep to be able to use extended regex symbols such as ?
* \"price\" matches the string "price"
* \s\* matches any number (including 0) of whitespace (\s) characters
* : matches :
* [0-9]\* matches any number of digits (from 0 to 9)
* ?\. optionally matches a . (this is in case price were an integer)

Now that you have the grep statement that you need, you can pipe the BTC data to it using the curl command from above:

$ curl -s --location --request GET https://api.coinstats.app/public/v1/coins/bitcoin\?currency\=USD |\

grep -oE "\"price\":\s\*[0-9]\*?\.[0-9]\*"

"price": 57907.78008618953

The backslash \ character used here after the pipe | allows you to write the expression on multiple lines.

Finally, to get *only* the value in the price field, and drop the "price" label, you can use chaining to pipe the same output to another grep:

$ curl -s --location --request GET https://api.coinstats.app/public/v1/coins/bitcoin\?currency\=USD |\

grep -oE "\"price\":\s\*[0-9]\*?\.[0-9]\*" |\

grep -oE "[0-9]\*?\.[0-9]\*"

57907.78008618953

Beautiful 😎

Linux and Bash Command Cheat Sheet: The Basics

Getting information

# return your user name  
whoami

# return your user and group id  
id

# return operating system name, username, and other info  
uname -a

# display reference manual for a command  
man top

# get help on a command  
curl --help

# return the current date and time  
date

Monitoring performance and status

# list selection of or all running processes and their PIDs  
ps  
ps -e

# display resource usage  
top

# list mounted file systems and usage  
df

Working with files

# copy a file  
cp file.txt new\_path/new\_name.txt

# change file name or path  
mv this\_file.txt that\_path/that\_file.txt

# remove a file verbosely  
rm this\_old\_file.txt -v

# create an empty file, or update existing file's timestamp  
touch a\_new\_file.txt

# change/modify file permissions to 'execute' for all users  
chmod +x my\_script.sh

# get count of lines, words, or characters in file  
wc -l table\_of\_data.csv  
wc -w my\_essay.txt  
wc -m some\_document.txt

# return lines matching a pattern from files matching a filename pattern - case insensitive and whole words only  
grep -iw hello \\*.txt

# return file names with lines matching the pattern 'hello' from files matching a filename pattern  
grep -l hello \\*.txt

Navigating and working with directories

# list files and directories by date, newest last  
ls -lrt

# find files in directory tree with suffix 'sh'  
find -name '\\*.sh'

# return present working directory  
pwd

# make a new directory  
mkdir new\_folder

# change the current directory: up one level, home, or some other path  
cd ../  
cd ~ or cd  
cd another\_directory

# remove directory, verbosely  
rmdir temp\_directory -v

Printing file and string contents

# print file contents  
cat my\_shell\_script.sh

# print file contents page-by-page  
more ReadMe.txt

# print first N lines of file  
head -10 data\_table.csv

# print last N lines of file  
tail -10 data\_table.csv

# print string or variable value  
echo "I am not a robot"  
echo "I am $USERNAME"

Compression and archiving

# archive a set of files  
tar -cvf my\_archive.tar.gz file1 file2 file3

# compress a set of files  
zip my\_zipped\_files.zip file1 file2  
zip my\_zipped\_folders.zip directory1 directory2

# extract files from a compressed zip archive  
unzip my\_zipped\_file.zip  
unzip my\_zipped\_file.zip -d extract\_to\_this\_direcory

Performing network operations

# print hostname  
hostname

# send packets to URL and print response  
ping www.google.com

# display or configure system network interfaces  
ifconfig  
ip

# display contents of file at a URL  
curl <url>

# download file from a URL  
wget <url>

Bash shebang

#!/bin/bash

Pipes and Filters

# chain filter commands using the pipe operator  
ls | sort -r

# pipe the output of manual page for ls to head to display the first 20 lines  
man ls | head -20

Shell and Environment Variables

# list all shell variables  
set

# define a shell variable called my\_planet and assign value Earth to it  
my\_planet=Earth

# display shell variable  
echo $my\_planet

# list all environment variables  
env

# environment vars: define/extend variable scope to child processes  
export my\_planet  
export my\_galaxy='Milky Way'

Metacharacters

# comments  
# The shell will not respond to this message

# command separator  
echo 'here are some files and folders'; ls

# file name expansion wildcard  
ls \*.json

# single character wildcard  
ls file\_2021-06-??.json

Quoting

# single quotes - interpret literally  
echo 'My home directory can be accessed by entering: echo $HOME'

# double quotes - interpret literally, but evaluate metacharacters  
echo "My home directory is $HOME"

# backslash - escape metacharacter interpretation  
echo "This dollar sign should render: \$"

I/O Redirection

# redirect output to file  
echo 'Write this text to file x' > x

# append output to file  
echo 'Add this line to file x' >> x

# redirect standard error to file  
bad\_command\_1 2> error.log

# append standard error to file  
bad\_command\_2 2>> error.log

# redirect file contents to standard input  
$ tr “[a-z]” “[A-Z]” < a\_text\_file.txt

# the input redirection above is equivalent to  
$cat a\_text\_file.txt | tr “[a-z]” “[A-Z]”

Command Substitution

# capture output of a command and echo its value  
THE\_PRESENT=$(date)  
echo "There is no time like $THE\_PRESENT"

Command line arguments

./My\_Bash\_Script.sh arg1 arg2 arg3

Batch vs. concurrent modes

# run commands sequentially  
start=$(date); ./MyBigScript.sh ; end=$(date)

# run commands in parallel  
./ETL\_chunk\_one\_on\_these\_nodes.sh & ./ETL\_chunk\_two\_on\_those\_nodes.sh

Scheduling jobs with Cron

# open crontab editor  
crontab -e

# job scheduling syntax  
m h dom mon dow command  
*minute, hour, day of month, month, day of week*  
\* means any

# append the date/time to file every Sunday at 6:15 pm  
15 18 \* \* 0 date >> sundays.txt

# run a shell script on the first minute of the first day of each month  
1 0 1 \* \* ./My\_Shell\_Script.sh

# back up your home directory every Monday at 3 am  
0 3 \* \* 1 tar -cvf my\_backup\_path\my\_archive.tar.gz $HOME\

# deploy your cron job  
*Close the crontab editor and save the file*

# list all cron jobs  
crontab -l

Shell Script: Conditionals

This reading will get you sufficiently familiar with bash *conditionals* for the final project.

Conditionals are ways of telling a script to do something *under specific condition(s)*.

In this reading, you will learn about shell script conditionals using if else.

If

**Syntax:**

**if** [ condition ]

**then**

statement

**fi**

You must always put spaces around your conditions in the [ ].

Every if condition block must be paired with a fi.

Example

$ cat if\_example.sh

a=1

b=2

**if** [ $a -lt $b ]

**then**

echo "a is less than b"

**fi**

$ sh if\_example.sh # sh tells the terminal to run the script if\_example.sh using the default shell

a is less than b

If-Else

**Syntax:**

**if** [ condition ]

**then**

statement\_1

**else**

statement\_2

**fi**

You don't use then for else cases.

Example

$ cat if\_else\_example.sh

a=3

b=2

**if** [ $a -lt $b ]

**then**

echo "a is less than b"

**else**

echo "a is greater than or equal to b"

**fi**

$ sh if\_else\_example.sh

a is greater than or equal to b

Elif

The statement elif means "else if":

**Syntax:**

**if** [ condition\_1 ]

**then**

statement\_1

**elif** [ condition\_2 ]

**then**

statement\_2

**fi**

Example

$ cat elif\_example.sh

a=2

b=2

**if** [ $a -lt $b ]

**then**

echo "a is less than b"

**elif** [ $a == $b ]

**then**

echo "a is equal to b"

**else** # Here a is not <= b, so a > b

echo "a is greater than b"

**fi**

$ sh elif\_example.sh

a is equal to b

Nested Ifs

As in other prgramming languages, it's also possible to nest if-statements.

**Syntax:**

**if** [ condition\_1 ]

**then**

statement\_1

**elif** [ condition\_2 ]

statement\_2

**if** [ condition\_2.1 ]

**then**

statement\_2.1

**fi**

**else**

statement\_3

**fi**

Example

$ cat nested\_ifs\_example.sh

a=3

b=3

c=3

**if** [ $a == $b ]

**then**

**if** [ $a == $c ]

**then**

**if** [ $b == $c ]

**then**

echo "a, b, and c are equal"

**fi**

**fi**

**else**

echo "the three variables are not equal"

**fi**

$ sh nested\_ifs\_example.sh

a, b, and c are equal

Alternatively, this example could have been simplified to a single if-statement:

a=3

b=3

c=3

**if** [ $a == $b ] && [ $a == $c ] && [ $b == $c ]

**then**

echo "a, b, and c are equal"

**else**

echo "the three variables are not equal"

**fi**

&& means "and"

Bonus: "test"

Sometimes, instead of using brackets around conditions, you'll see the test command in use:

Example

$ cat test\_example.sh

a=1

b=2

**if** test $a -lt $b

**then**

echo "a is less than b"

**fi**

$ sh test\_example.sh

a is less than b

test and [ ] are the same command. We encourage using [ ] instead as it's more readable.