ALX LESSON

0x02. Shell, IO Redirections and filters

Shell - Bash

TABLE OF CONTENTS

01

OVERVIEW topics

Ubuntu Shell Commands

Topics

how to redirect standard output to a file

how to get standard input from a file instead of keyboard

how to send output from one program to the input of another program

how to combine commands and filters with redirections

what are spectial characters

understand what do the white spaces, single qoutes, double qoutes, backslash, comment, pipe, command separator, tilde and how and when to use them

Ubuntu Shell Commands

Topics

how to display a line of text

how to concatenate files and print on the standard output

how to reverse a string

how to remove sections from each line of lines

Ubuntu Shell Commands

Topics

echo
cat
wc
head
tail
sort
uniq
tr
grep
cut
sed
awk
tee
find

Slides On Telegram

https://t.me/alx_2023

Ubuntu Shell Commands

Topics

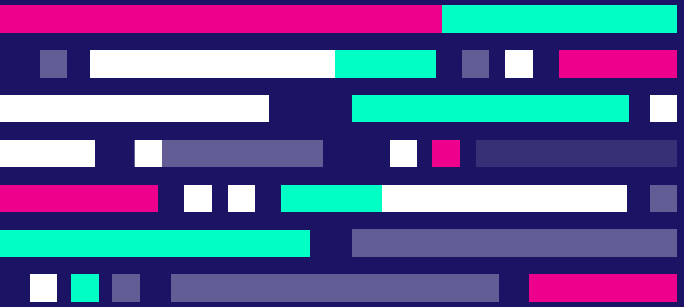
02

Learning Objectives

echo

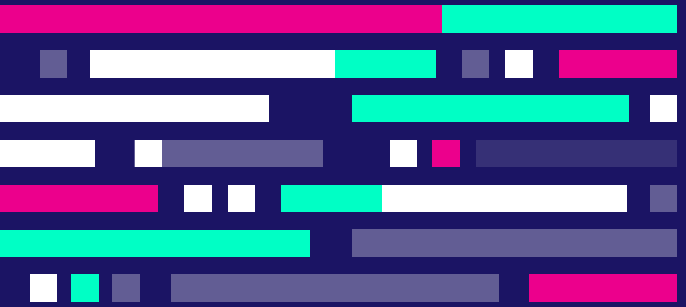
Parameter	Description	Example
`-n`	Suppresses the trailing newline character	`echo -n "Hello world"` prints "Hello world" without a newline at the end.
`-e`	Enables the interpretation of escape sequences	`echo -e "Hello\tworld"` prints "Hello" followed by a tab character and then "world".
`-E`	Disables the interpretation of escape sequences	`echo -E "Hello\tworld"` prints "Hello\tworld" without interpreting the tab character.
`-\\`	Escapes the next character and treats it as a literal character	`echo "Hello \\world"` prints "Hello \world".
`-`	Treats all subsequent arguments as positional parameters	`echo - "Hello world"` prints "Hello world" and any subsequent arguments as separate positional parameters.
`--`	Indicates the end of options	`echo -- -Hello` prints "-Hello" and treats it as a string, rather than an option.
`-help`	Displays help information for the `echo` command	`echo --help` displays help information for the `echo` command.

echo



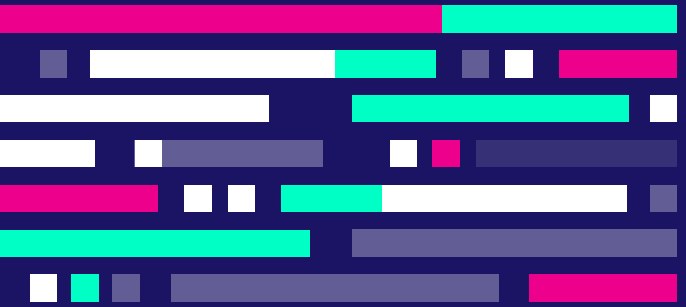
```
echo "hello world!"

echo "my bank balance is: $500"
echo "my bank balance is: \$500"

echo $PWD
echo "The current date and time is $(date)"
echo number_{1..5}

echo front-{A,B,C}-Back
echo front-{A..C}-Back

echo {A..Z}

echo *

echo "Hello\tworld\nWelcome to\n\nUnix/Linux"
echo -e "Hello\tworld\nWelcome to\n\nUnix/Linux"
```

The characters that need to be escaped with a backslash \



Backslash:	\
Single quote:	'
Double quote:	"
Dollar sign:	$
Ampersand:	&
Asterisk:	*
Question mark:	?
Parentheses:	( )
Brackets:	[ ]
Braces:	{ }
Pipe symbol:	|
Less-than symbol:	<
Greater-than symbol:	>
Semi-colon:	;
Colon:	:

The characters that need to be interpreter -e



Sequence	Character printed
\a	Alert or Ctrl-G (bell)
\b	Backspace or Ctrl-H
\c	Omit final newline
\e	Escape character (same as \E)
\E	Escape character
\f	Formfeed or Ctrl-L
\n	Newline (not at end of command) or Ctrl-J

`\r`	Return (Enter) or Ctrl-M
`\t`	Tab or Ctrl-I
`\v`	Vertical Tab or Ctrl-K
`\n` *nnn*	The eight-bit character whose value is the octal (base-8) value *nnn* where *nnn* is 1 to 3 digits
`\0` *nnn*	The eight-bit character whose value is the octal (base-8) value *nnn* where *nnn* is 0 to 3 digits
`\x`*HH*	The eight-bit character whose value is the hexadecimal (base-16) value *HH* (one or two digits)
`\\`	Single backslash

shell redirection operators

Operator	Description	Example
`<`	Input Redirection: read input from a file or command	`command < input.txt`
`>`	Output Redirection: write output to a file or device	`command > output.txt`
`>>`	Append Output Redirection: append output to a file	`command >> output.txt`
`2>`	Error Redirection: write error output to a file	`command 2> error.txt`
`&<`	Input/Output Redirection: read input and write output	`command &< input_output.txt`
`&>`	Input/Output Redirection: write output and error output	`command &> output_error.txt`
`<>`	Input/Output Redirection: read input and write output	`command <> input_output.txt`
`<<`	Here Document: redirect multiple lines of input to a command	`command << END`
`<<<`	Here String: redirect a string as input to a command	`command <<< "hello"`

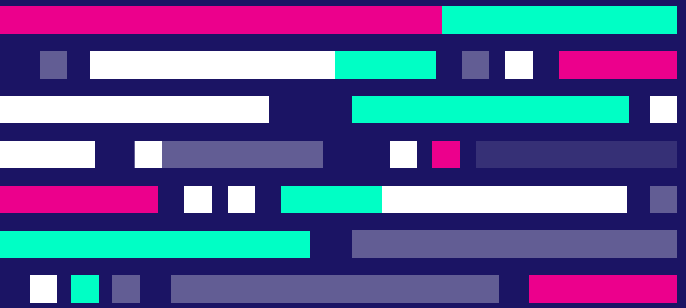
shell redirection operators

File	File Descriptor
Standard Input STDIN	0
Standard Output STDOUT	1
Standard Error STDERR	2

```
echo "hello, world!" > outputfile.txt
echo "again, hello" >> outputfile.txt

wc < outputfile.txt
wc << outputfile.txt
cat <<< outputfile.txt

ls /path/to/missing/directory 2> error.txt
ls /path/to/nonexistent/directory &> output_and_error.txt
ls {.,/path/to/nonexistent/directory} &> output_and_error.txt
ls {.,/path/to/nonexistent/directory} > output_and_error.txt
2>&1
```

shell redirection operators

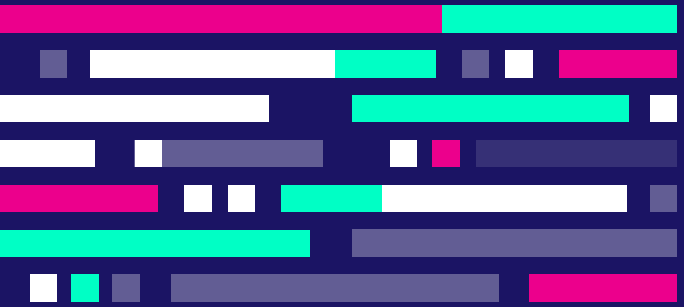
`&`	Runs a command in the background
`;`	Separates multiple commands on a single command line, runs them sequentially
`&&`	Separates multiple commands on a single command line, runs them conditionally (only runs the next command if the previous command succeeds)



The | operator connects the output of one command to the input of another command, allowing for powerful command pipelines to be constructed.

```
ls -l | grep ".txt"
ls -l ; echo "done"
ls -l && echo "okay"
ls fredkorhan && echo "not okay“
echo Don’t work if the command before it works first
```

cat



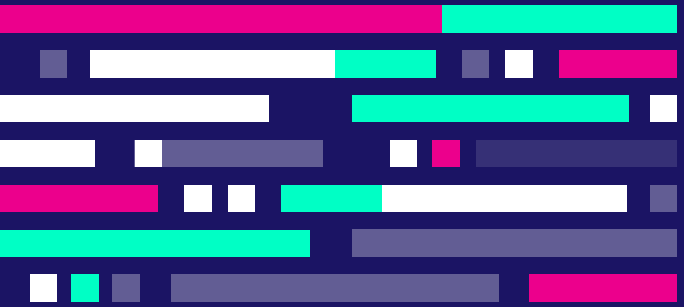
The cat command is a utility command in Linux. One of its most common usages is to print the content of a file onto the standard output stream. Other than that, the cat command also allows us to write some texts into a file.

Display the contents of a file:
`cat filename.txt`

Concatenate two or more files and display the output:
`cat file1.txt file2.txt`

Append the contents of one file to another file:
`cat file1.txt >> file2.txt`

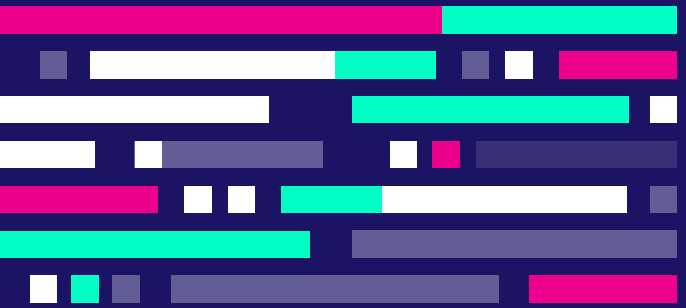
cat



Create a new file and add content to it:
`cat > newfile.txt`

This will create a new file called newfile.txt and allow you to type in content. Press Ctrl-D to save and exit.

wc



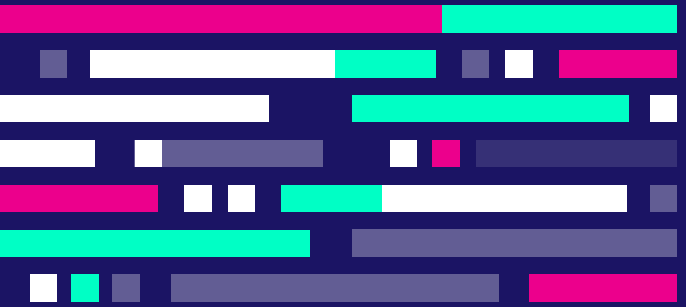
The wc command in Linux is a command-line utility used to count the number of lines, words, and characters in a file or input from the standard input.

Count the number of lines, words, and characters in a file:
`wc filename.txt`

Count the number of lines, words, and characters in multiple files:
`wc file1.txt file2.txt`

Count the number of lines, words, and characters in the output of another command:
`ls | wc`

wc

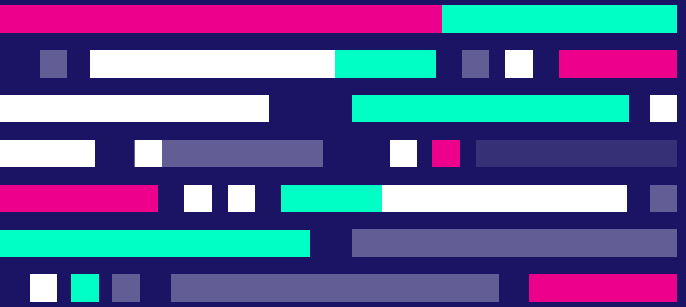


Options:

```
-c, --bytes            print the byte counts
-m, --chars            print the character counts
-l, --lines            print the newline counts
-L, --max-line-length  print the maximum display width
-w, --words            print the word counts
```

```
wc -m filename.txt
wc -l filename.txt
wc -w filename.txt
wc -L filename.txt
```

head



The head command in Linux is a command-line utility used to display the first few lines of a file or input from the standard input.

This will display the first 10 lines of filename.txt by default:

```
head filename.txt
```

Display a specified number of lines from a file:

```
head -n 15 filename.txt
```

This will display the first 10 lines of file1.txt and file2.txt by default.

```
head file1.txt file2.txt
```

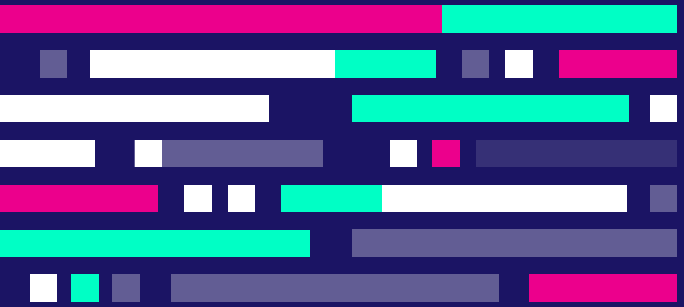
To display the first 50 bytes of a file, you would use the following command:

```
head -c 50 filename.txt
```

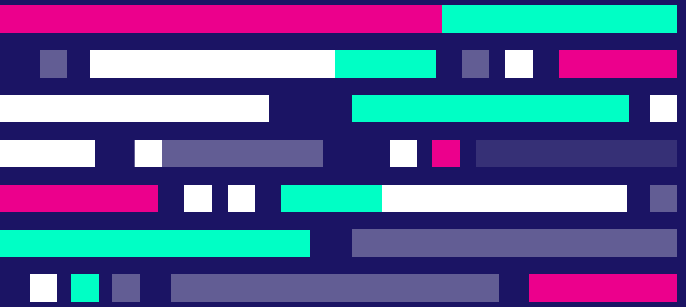
head

To display the file name before each output line

```
head -v filename.txt
```



tail



The tail command is used to display the last few lines of a file. Here are some examples of how to use tail:

```
tail file.txt
```

This command will display the last 5 lines of the file file.txt

```
tail -n 5 file.txt
```

This command will display the last 10 lines of the file log.txt, and then keep the file open and display any new lines that are added to the file in real time

```
tail -f log.txt
```

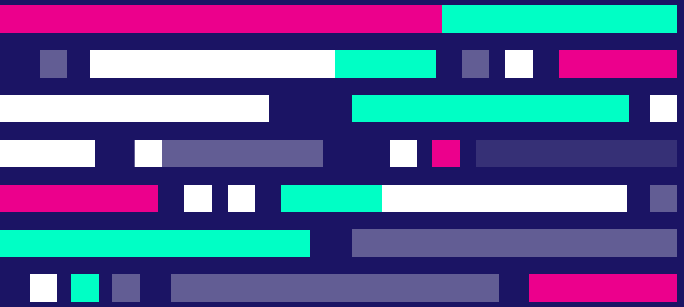
This command will display the last 1000 bytes of the file file.txt

```
tail -c 1000 file.txt
```

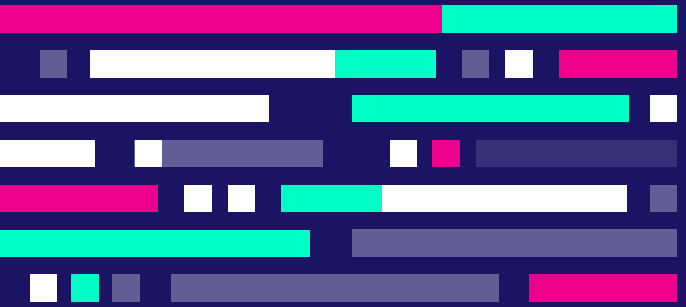
tail

This command will display all lines in file.txt to the fifth line.

```
tail -n +5 file.txt
```



sort



The sort command is used to sort lines of text files in alphabetical or numerical order. By default, it sorts lines in ascending order.

This command will sort the lines in file.txt in alphabetical order and print the sorted lines to the terminal.

```
sort file.txt
```

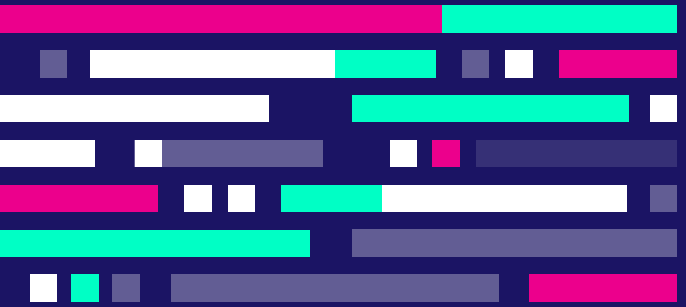
This command will sort the lines in file.txt in reverse alphabetical order and print the sorted lines to the terminal.

```
sort -r file.txt
```

This command will sort the lines in numbers.txt in numerical order and print the sorted lines to the terminal.

```
sort -n numbers.txt
```

sort



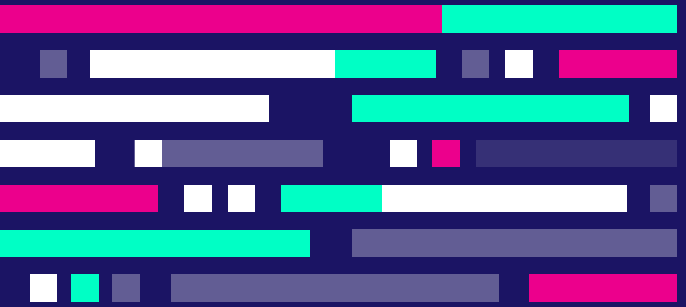
This command will sort the lines in numbers.txt in reverse numerical order and print the sorted lines to the terminal.

```
sort -nr numbers.txt
```

This command will sort the lines in file.txt in alphabetical order and remove any duplicates, printing only the unique lines to the terminal.

```
sort -u file.txt
```

uniq



The uniq command is used to identify and remove duplicates from a sorted text file or input stream. By default, it compares adjacent lines in the input and removes any duplicates.

This command will read file.txt and remove any duplicate lines, printing only the unique lines to the terminal.

```
uniq file.txt
```

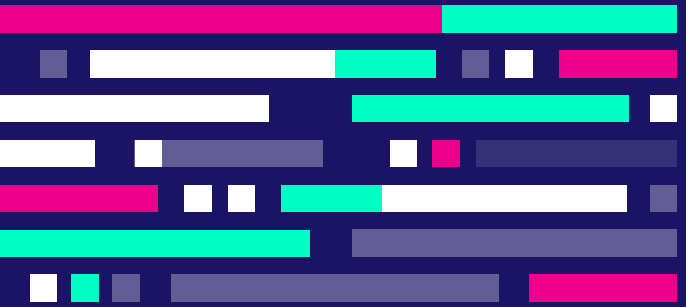
This command will read file.txt and count the number of occurrences of each unique line, printing the count and the line to the terminal.

```
uniq -c file.txt
```

This command will read file.txt and print only the lines that appear more than once (i.e., the duplicate lines).

```
uniq -d file.txt
```

uniq

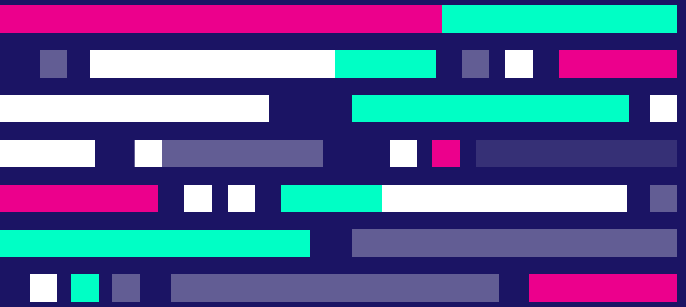


This command will read file.txt and print only the lines that appear exactly once (i.e., the unique lines).

```
uniq -u file.txt
```

```
sort file.txt | uniq
```

tr



The tr command in Linux is used to translate or delete characters. It can be used to replace or remove characters from a text file or input stream.

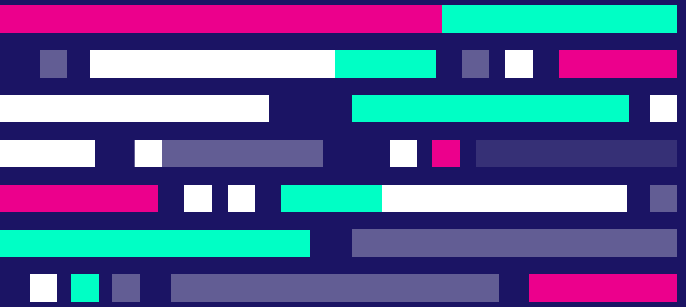
This command will replace all occurrences of the letter "o" with the number "0" in the input string "hello world", and print the result to the terminal:

```
echo "hello world" | tr "o" "0“
```

This command will read the contents of file.txt and convert all lowercase letters to uppercase letters using the [:lower:] and [:upper:] character classes, and then print the result to the terminal.

```
cat file.txt | tr "[:lower:]" "[:upper:]"
```

tr



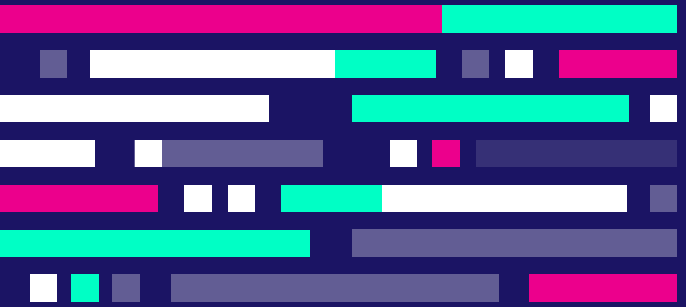
This command will read the contents of file.txt and remove all punctuation marks (such as periods, commas, and semicolons) using the [:punct:] character class, and then print the result to the terminal.

```
cat file.txt | tr -d "[:punct:]"
```

This command will replace all consecutive spaces in the input string "hello world" with a single space using the -s option, and print the result to the terminal:

```
echo "hello    world" | tr -s " "
```

grep



This command will search the file file.txt for any lines that contain the word "hello", and print those lines to the terminal.

```
grep "hello" file.txt
```

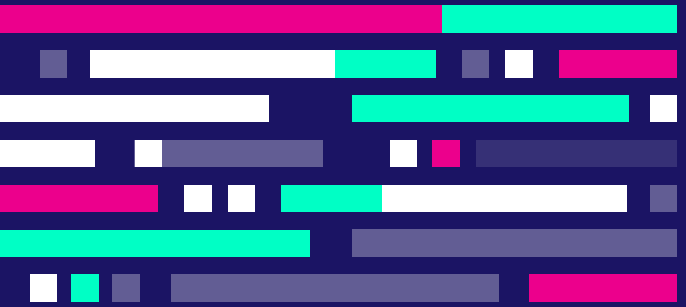
This command will search the file file.txt for any lines that contain the word "hello", regardless of the case (since we're using the -i option), and print those lines to the terminal.

```
grep -i "hello" file.txt
```

This command will search recursively through the directory directory/ for any files that contain the word "hello", and print the names of those files to the terminal.

```
grep -r "hello" directory/
```

grep



This command will search the file file.txt for any lines that begin with the word "hello" (since we're using the ^ character to anchor the search to the beginning of the line), and print those lines to the terminal.

```
grep "^hello" file.txt
```

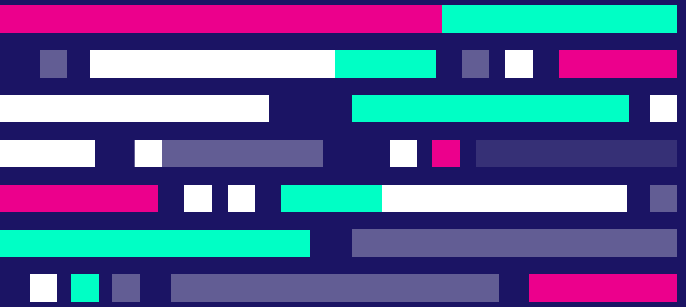
This command will list all running processes using the ps command, and then filter the results to show only the ones that contain the word "chrome" (in this case, we're using grep to search the output of the ps command).

```
ps -ef | grep "chrome"
```

This command will print all lines in file.txt that do not contain the word "hello".

```
grep -v "hello" file.txt
```

grep



This command will search all files in the current directory for the word "hello", and print the names of the files that contain it.

```
grep -l "hello" *
```

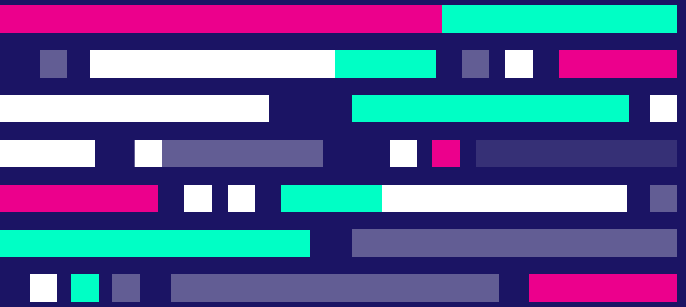
This command will print the number of lines in file.txt that contain the word "hello".

```
grep -c "hello" file.txt
```

This command will print all lines in file.txt that contain the whole word "hello", but not lines that contain partial matches like "hello there" or "helloworld".

```
grep -w "hello" file.txt
```

grep



-E This option allows you to use extended regular expressions:

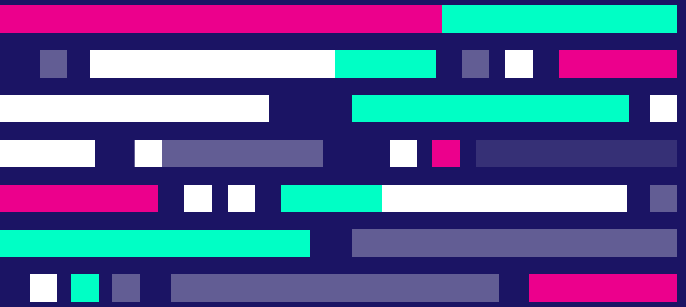
grep -E "hello|world" file.txt

This command will print all lines in file.txt that contain either the word "hello" or the word "world", using the | character to separate the two options.

grep -A 1 "lazy" file.txt

the -A 1 option tells grep to print one line after each match of the pattern "lazy"

cut



The cut command in Linux is used to extract specific columns or fields from a file or input stream. It can be used to split a file into parts based on a delimiter or a fixed width.

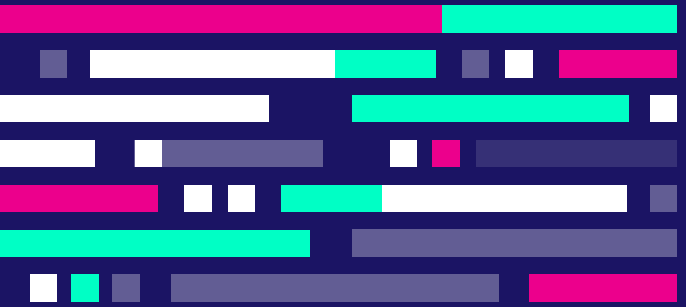
cut is used to extract the first five characters from each line of the file.txt.

```
cut -c 1-5 file.txt
```

cut is used to extract the first and third fields from each line of the file.csv, which are separated by commas.

```
cut -d , -f 1,3 file.csv
```

sed



stands for "stream editor" and is a powerful tool for text manipulation in Linux and Unix-based systems. It can be used to perform a wide variety of operations on text files and streams, such as searching, replacing, inserting, and deleting text.

Here, s is the sed command to search and replace text, and test and sample are the search string and replacement string, respectively.

```
sed 's/test/sample/' example.txt
```

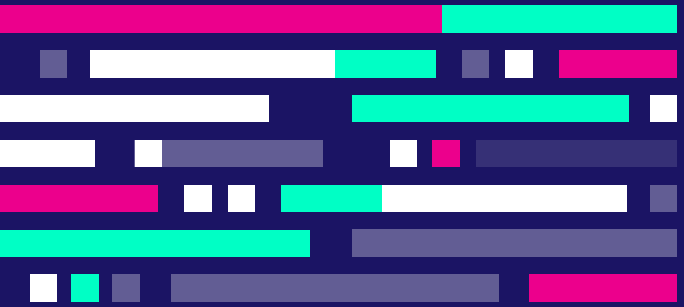
Here, 2d is the sed command to delete the second line.

```
sed '2d' example.txt
```

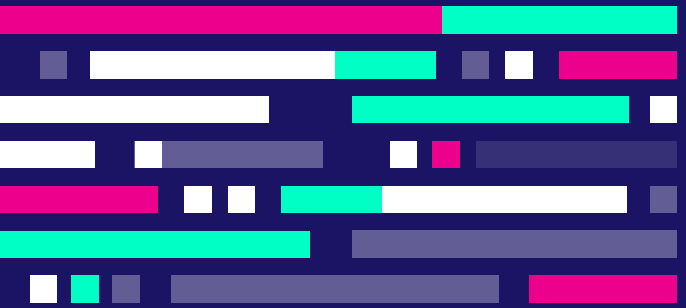
sed

```
sed ‘1a This is a new line.' example.txt
```

Here, 1a is the sed command to append text after the first line, and the backslash and newline are used to separate the command from the text to insert.



awk



Awk is a powerful text processing tool that is commonly used in Unix and Linux environments. It allows you to search for and manipulate patterns in text data, making it a useful tool for a wide range of tasks.

```
awk options 'pattern { action }' filename
```

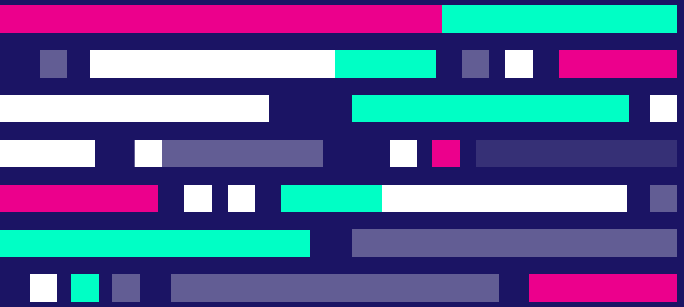
```
awk -F',' '{ print $1 }' data.txt
```

Here, the -F option specifies the field separator as a comma. The '{ print $1 }' action tells awk to print the first field (i.e., the fruit names) for each line of input.

```
awk -F'[.,]' '{print $1}' prices.txt
```

This command uses a regular expression to split each line of the prices.txt file into fields using either a . or , as the field separator.

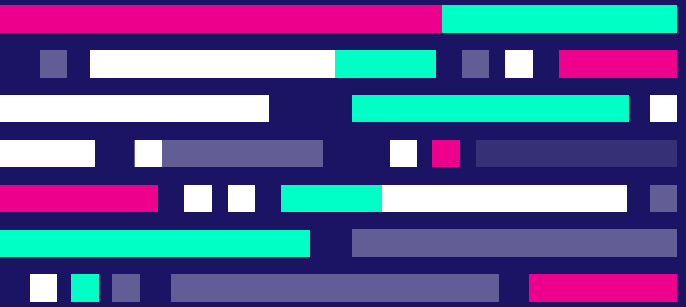
awk



```
awk -F':' '{print $1,$7}' /etc/passwd
```

This command prints the first and seventh fields (username anda shell) of each line in the /etc/passwd file, using : as the field separator and a space as the output field separator.

tee



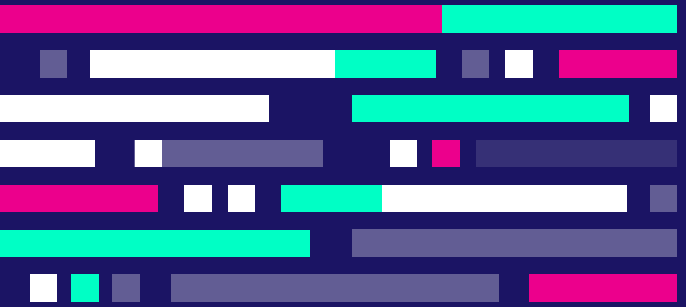
The tee command is used to read the standard input and then write it to both the standard output and one or more files.

ls -l | tee file.txt
This command will write the output of the ls -l command to the file file.txt and also display it on the terminal.

echo "New line" | tee -a file.txt
This command will append the string "New line" to the end of the file file.txt.

echo "Hello world" | tee file1.txt file2.txt
This command will write the string "Hello world" to both file1.txt and file2.txt.

tee

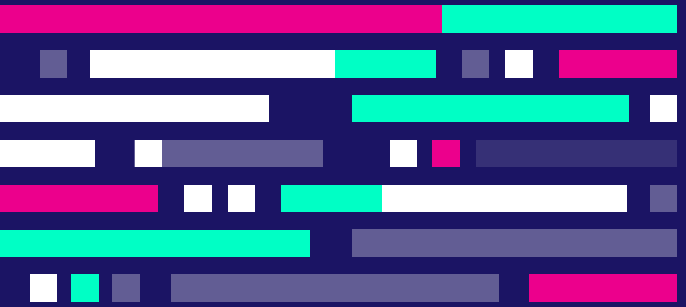


```
cat bigfile.txt | tee -i output.txt
```

This command will read the contents of bigfile.txt and write them to output.txt, ignoring any interrupt signals that may be sent to the tee command.

-i or --ignore-interrupts: This option tells tee to ignore interrupt signals like CTRL+C, allowing it to continue running even if the user tries to stop it.

find



The find command is a powerful tool for searching for files in a directory hierarchy.

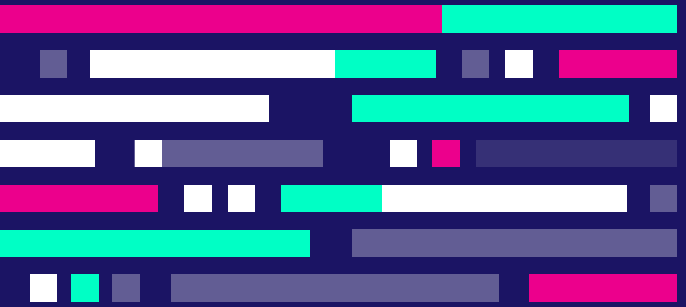
Find all files in the current directory and its subdirectories:
`find .`

Find all directories in the current directory and its subdirectories:
`find . -type d`

Find all files in the current directory and its subdirectories that have the extension ".txt":
`find . -name "*.txt"`
`find . -iname “FILE.txt"` Similar to -name, but performs a case-insensitive search.

Find all files in the current directory and its subdirectories that were modified more than 30 days ago:
`find . -mtime +30`

find



Find all files in the current directory and its subdirectories that are larger than 1 megabyte:

```
find . -size +1M
```

Find all files in the current directory and its subdirectories that have not been accessed in the last 7 days:

```
find . -atime +7
```

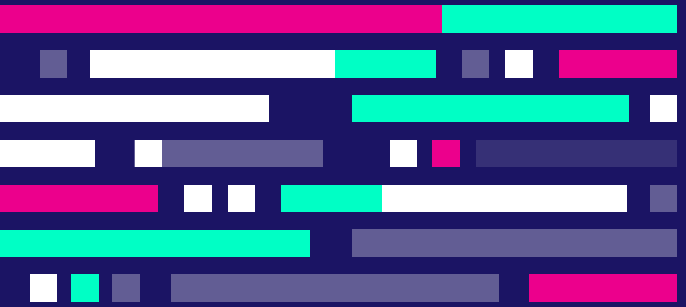
Find all files in the current directory and its subdirectories that are owned by the user "john":

```
find . -user john
```

Find all files in the current directory and its subdirectories that are not owned by the current user:

```
find . ! -user $(whoami)
```

find



Find all files in the current directory and its subdirectories that are executable by the owner:

```
find . -perm /u+x
```

Find all files in the current directory and its subdirectories that are writable by anyone:

```
find . -perm /o+w
```

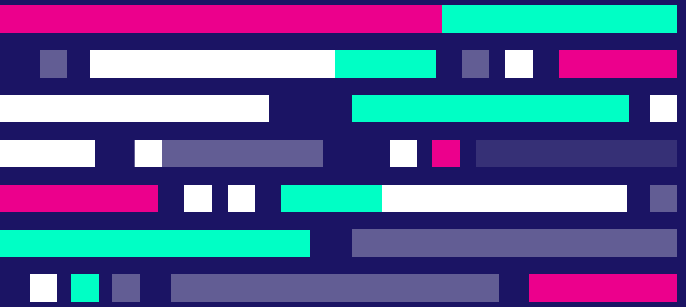
```
find . -mindepth 1 -type f
```

This will list all regular files (not directories or symbolic links) in the current directory and its subdirectories.

```
find . -mindepth 2 -type f
```

This will list all regular files not in the current directory but in its subdirectories.

find



`find . -maxdepth 1 -type f`
This will list all regular files (not directories or symbolic links) in the current directory and its subdirectories.

`find . -maxdepth 2 -type f`
This will list all regular files in the current directory and its subdirectories.

`find . -name "*.txt" -exec cat {} \;`
This command will find all files in the directory that end with .txt and execute the cat command on each of them.

`find . -type f -name "*.txt" -print`
This command will find all files in the directory that end with .txt and print their names to the terminal.

04

Hands on lab Practice


Have a Question
Leave a Comment!

Share

To let the others know more

Subscribe

To stay updated with latest videos

Thanks