Text Processing

Hadeer Ramadan

Table of contents

01

Text manipulation commands

Cut, Sort, uniq

02

grep

Searching and Pattern Matching

03

Regex

Text Processing

whether you're managing servers, analyzing data, or maintaining complex configurations across multiple environments. The command-line approach offers unmatched speed, flexibility, and power, making it a key skill for anyone working in technical fields.

Efficiency

Efficient text processing helps in extracting valuable information quickly

Data Analysis

Processing logs and configuration files aids in monitoring troubleshooting, and performance tuning.

Customization

Customizing outputs and generating reports tailored to specific needs

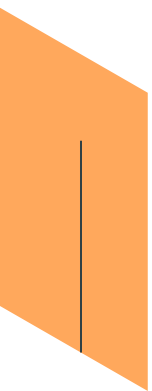
01

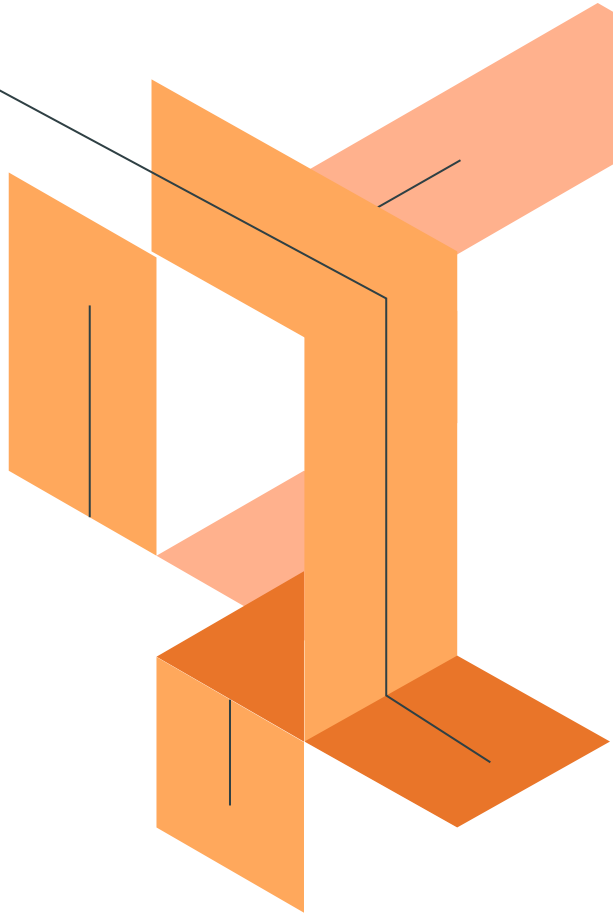
Text Manipulation Commands

Cut

- Cut The cut program is used to extract a section of text from aline and output the extracted section to standard output

- Command syntax:
- `cut option file



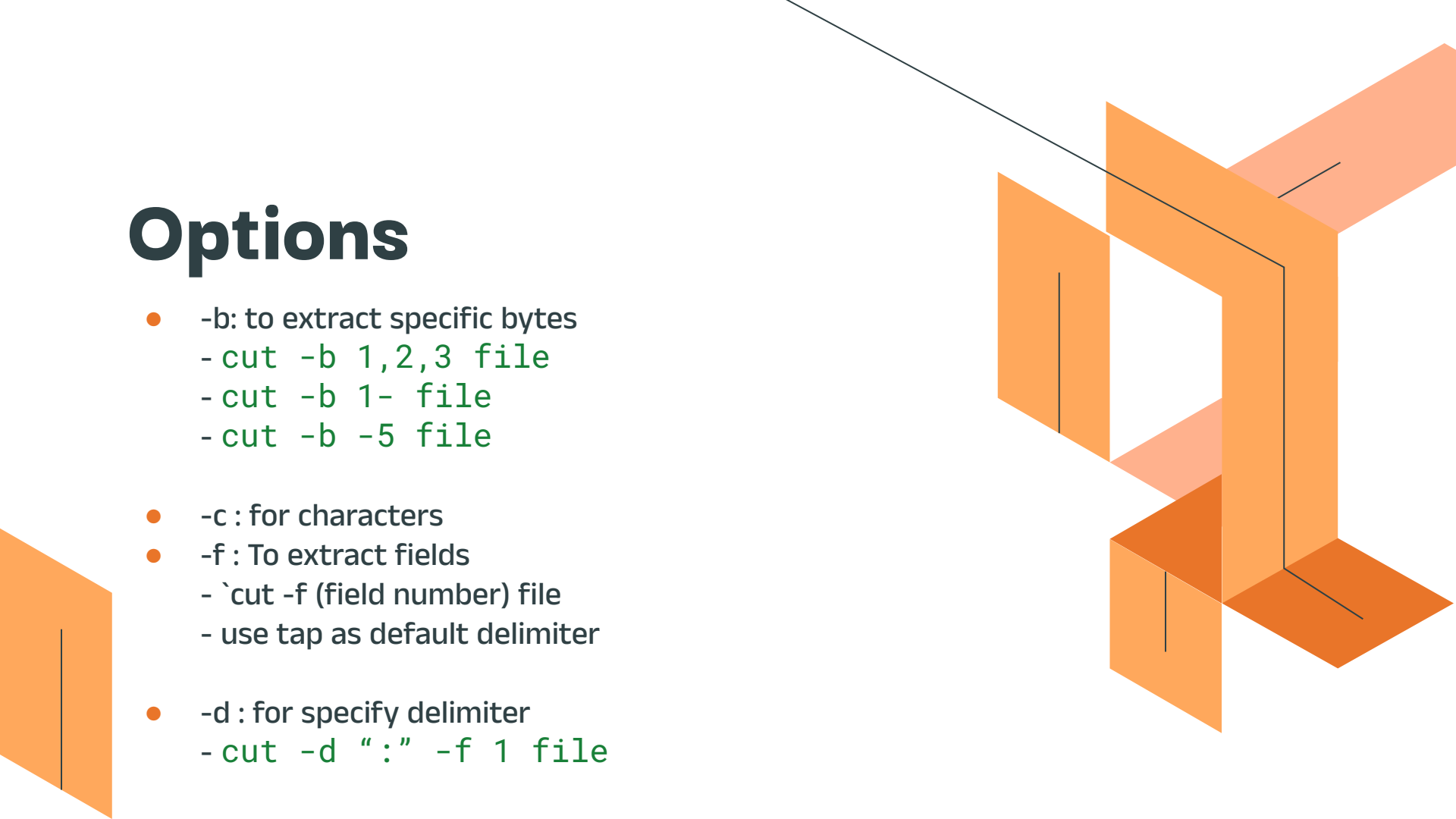


Options

- -b: to extract specific bytes
  - `cut -b 1,2,3 file`
  - `cut -b 1- file`
  - `cut -b -5 file`

- -c : for characters
- -f : To extract fields
  - `cut -f (field number) file
  - use tap as default delimiter

- -d : for specify delimiter
  - `cut -d ":" -f 1 file`



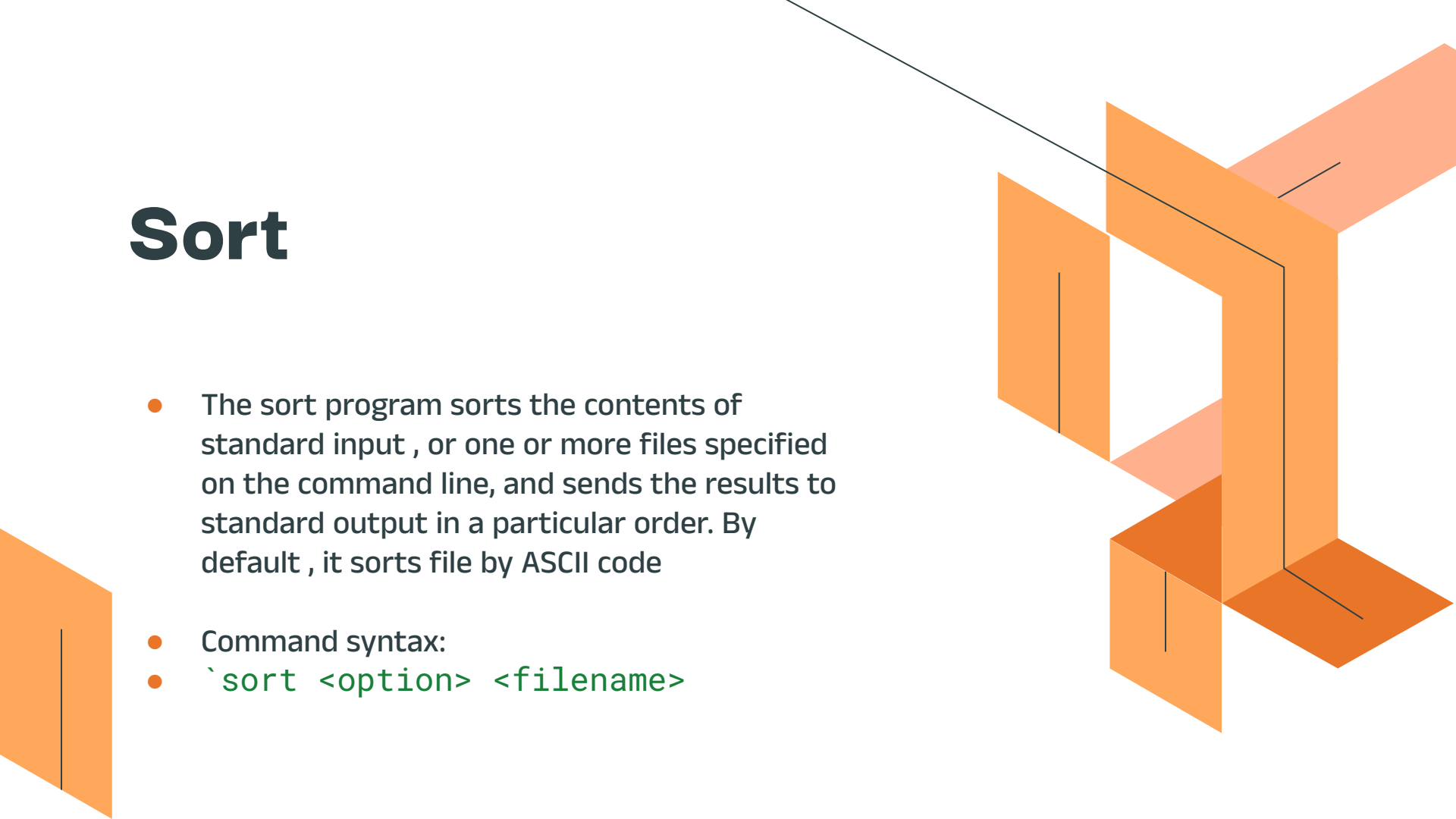
Examples

```
hadeer@hadeer-ASUS:~$ cat feilds
root    x       0
daemon  x       1
bin     x       2
sys     x       3
adm     x       4       syslog,hadeer
tty     x       5
disk    x       6
lp      x       7
hadeer@hadeer-ASUS:~$ cut -f 1 feilds
root
daemon
bin
sys
adm
tty
disk
lp
```

Sort

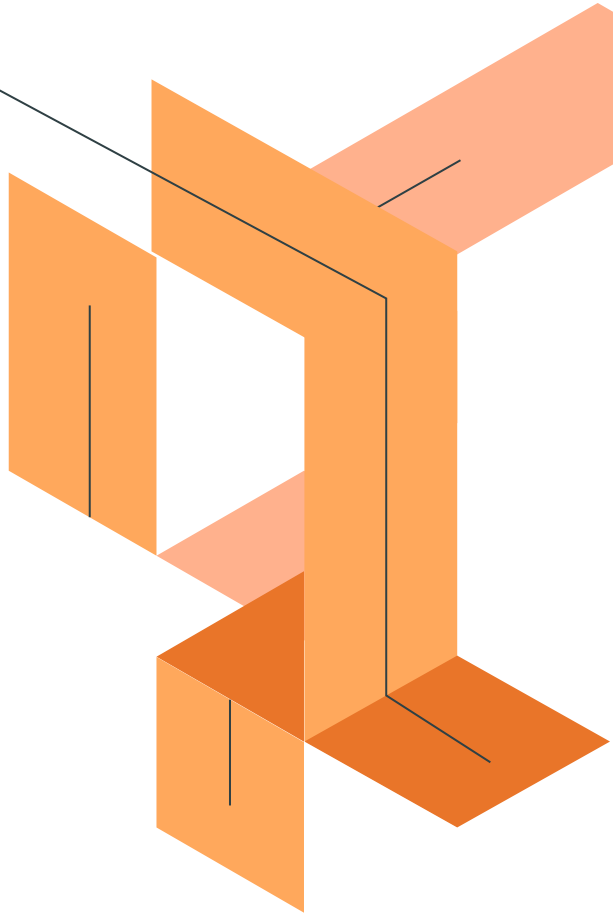
- The sort program sorts the contents of standard input , or one or more files specified on the command line, and sends the results to standard output in a particular order. By default , it sorts file by ASCII code

- Command syntax:
- `` `sort <option> <filename> ``



Options

-c	to check if the file given is already sorted or not
-n	sort based on numeric values
-r	reverse sorting
-k	sorting a file according to any given field number
-t	Define the field-separator character.



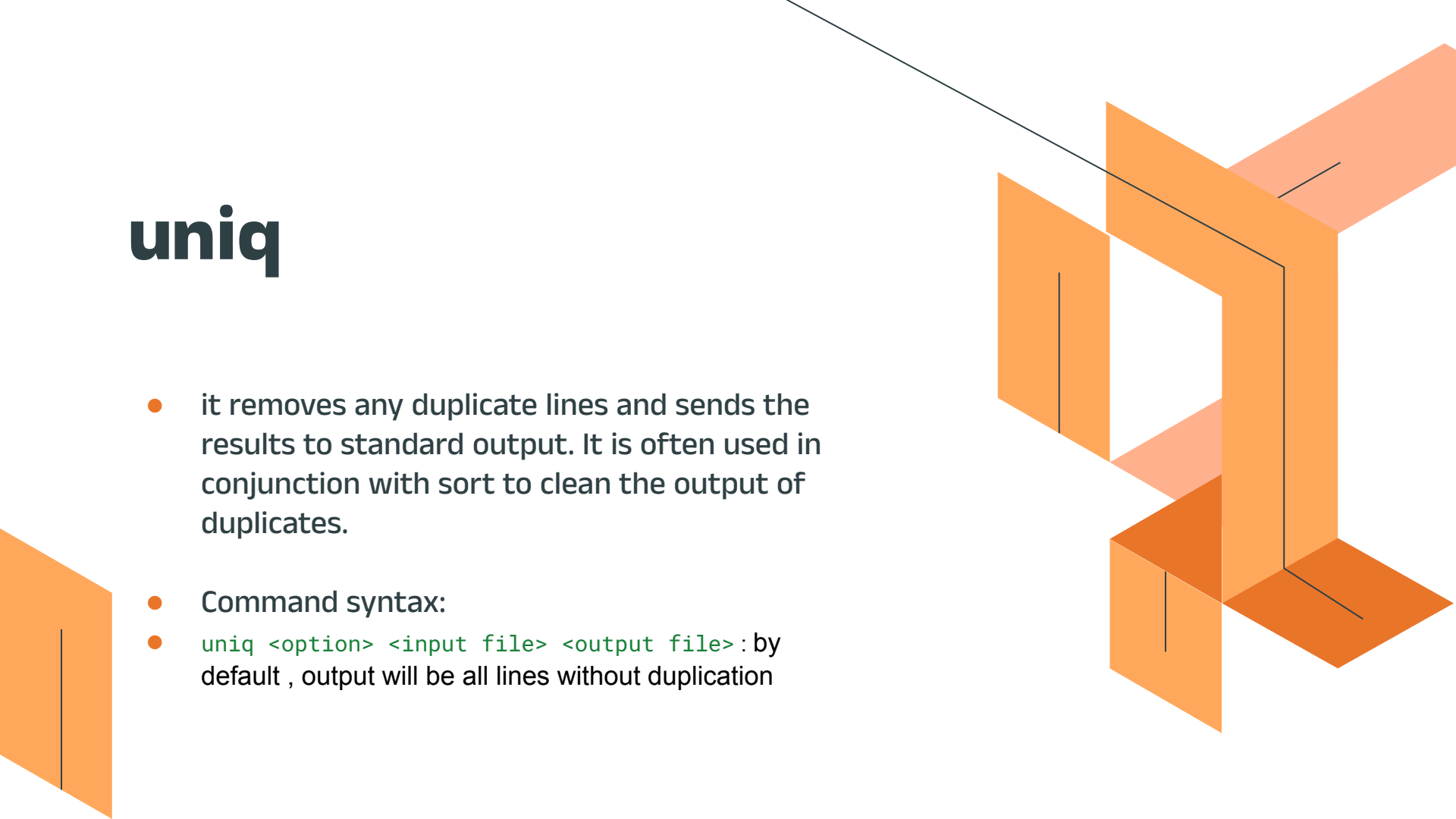
Examples

```
hadeer@hadeer-ASUS:~$ sort -t ":" -k 3 feilds
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,hadeer
tty:x:5:
disk:x:6:
```

uniq

- it removes any duplicate lines and sends the results to standard output. It is often used in conjunction with sort to clean the output of duplicates.

- Command syntax:
- `uniq <option> <input file> <output file>` : by default , output will be all lines without duplication



Options

-c	print each output line by the number of times it occur
-d	Display the repeated lines
-u	Display the lines that are not repeated
-i	Ignore case sensitive

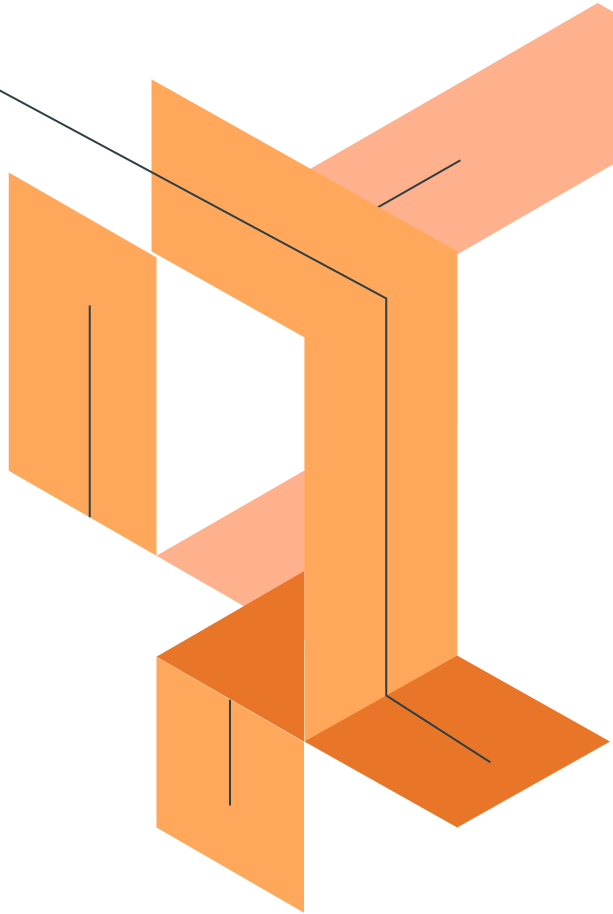
02

Searching and Pattern matching

grep: searching in text

- “Grep” stands for ‘Global Regular expression print’.
- used to search files for the occurrence of a string of characters that matches a specified pattern

- Command syntax:
- `grep <options> <text> <file>`



Options

-l	Print name of each file that contain match text
-n	Print number of line with line that matches tex
-r	recursive searching in directory
-i	ignore case sensitive
-v	print all lines that doesn't match text
-c	Print the number of matches
-A	Print num lines of trailing context after matching lines.
-B	Print num lines of leading context before matching lines.

03

Regex

Reviewing concepts is a good idea

- Regular Expressions are special characters which help search data and matching complex patterns. Regular expressions are shortened as ‘regexp’ or ‘regex’. They are used in many Linux programs like grep, bash, sed, etc.

Versions

1. BRE : Basic Regular Expression
2. ERE : Extend Regular Expression
3. PCRE : Perl-compatible Expression

Types of Special characters

Metacharacters that are special by default:

- `.^$*[]\`

Characters that become special when escaped by backslash `\`:

- `?+{}|()`

Symbol Description

1. Anchors

^	matches start of string
$	matches end of string

2. Backslash and Alteration

\\	Matches literal backslash.
|	using as OR, so “aa|bb” matches aa or bb.

3. Repetitions

.	matches only one single character
?	matches zero or one for the preceding item
*	matches zero or more for the preceding item
+	matches one or more for the preceding item
\{n\}	matches exactly n times for the preceding item
\{n,\}	matches at least n times for the preceding item
\{,m\}	matches at most m times for the preceding item
\{n,m\}	matches from n to m times for the preceding item

? and +

```
hadeer@hadeer-ASUS:~/textProcessing$ grep "e\?" deep
dp dep deep
hadeer@hadeer-ASUS:~/textProcessing$ grep "e\+" deep
dp dep deep
hadeer@hadeer-ASUS:~/textProcessing$ grep -o  "e\+" deep
e
ee
hadeer@hadeer-ASUS:~/textProcessing$ grep -o  "e\?" deep
e
e
e
```

\{\}

```
hadeer@hadeer-ASUS:~/textProcessing$ cat fruits
Apple
Banana
Mango
Orange
Pineapple
Strawberry
hadeer@hadeer-ASUS:~/textProcessing$ grep "p\{1\}" fruits
Apple
Pineapple
hadeer@hadeer-ASUS:~/textProcessing$ grep -o "p\{1\}" fruits
p
p
p
p
hadeer@hadeer-ASUS:~/textProcessing$ grep -o "p\{2,\}" fruits
pp
pp
```

4. Character Classes and Bracket Expressions

alnum	alphanumeric characters (letters and numbers)
alpha	alphabetic characters (letters)
digit	digits (numbers)
lower	uppercase letters
upper	punctuation characters
punct	punctuation characters
print	Printable Characters , includes alnum, punt , space
space	Space characters

Class name & invert match

```
hadeer@hadeer-ASUS:~/textProcessing/data$ grep "[[:digit:]]" info
John 30
Jane 25
Michael 50
Emily 20
David 11
hadeer@hadeer-ASUS:~/textProcessing/data$ grep "[^[:digit:]]" info
John 30
Jane 25
Michael 50
Emily 20
David 11
Sarah Johnson
```

Ranges & invert match

```
hadeer@hadeer-ASUS:~/textProcessing/data$ grep "[1-3]" info
John 30
Jane 25
Emily 20
David 11
hadeer@hadeer-ASUS:~/textProcessing/data$ grep "[^1-3]" info
John 30
Jane 25
Michael 50
Emily 20
David 11
Sarah Johnson
```

Revert class name or match

- By adding ^ before pattern
- `grep “[^[:digit:]]” info`
- `Grep “[^1-3]” info`

Grouping and Capturing: \(\)

Grouping: \(text\)

- example : `grep “\(abc\)\{2\}” file`

Capturing: it capture a pattern, and in first match it save match in memory and specify number to refer to this match. So when you need to call or check for this exactly match again just call it by \num. numbers are arranged from 1 (1,2,3,..etc). In simple way you can check for duplicates words by using capture pattern.

Thanks!