Paste command in Linux with examples

Paste command is one of the useful commands in Unix or Linux operating system. It is used to join files horizontally (parallel merging) by outputting lines consisting of lines from each file specified, separated by **tab** as delimiter, to the standard output. When no file is specified, or put dash (“-“) instead of file name, paste reads from standard input and gives output as it is until a interrupt command **[Ctrl-c]** is given.

**Syntax:**

**paste [OPTION]... [FILES]...**

Let us consider three files having name **state**, **capital** and **number**. *state* and *capital* file contains 5 names of the Indian states and capitals respectively. *number* file contains 5 numbers.

**$ cat state**

Arunachal Pradesh

Assam

Andhra Pradesh

Bihar

Chhattisgrah

**$ cat capital**

Itanagar

Dispur

Hyderabad

Patna

Raipur

Without any option paste merges the files in parallel. The paste command writes corresponding lines from the files with tab as a deliminator on the terminal.

**$ paste number state capital**

1 Arunachal Pradesh Itanagar

2 Assam Dispur

3 Andhra Pradesh Hyderabad

4 Bihar Patna

5 Chhattisgrah Raipur

In the above command three files are merges by paste command.

**Options:**

**1. -d (delimiter):** Paste command uses the tab delimiter by default for merging the files. The delimiter can be changed to any other character by using the **-d** option. If more than one character is specified as delimiter then paste uses it in a circular fashion for each file line separation.

**Only one character is specified**

**$ paste -d "|" number state capital**

1|Arunachal Pradesh|Itanagar

2|Assam|Dispur

3|Andhra Pradesh|Hyderabad

4|Bihar|Patna

5|Chhattisgrah|Raipur

**More than one character is specified**

**$ paste -d "|," number state capital**

1|Arunachal Pradesh,Itanagar

2|Assam,Dispur

3|Andhra Pradesh,Hyderabad

4|Bihar,Patna

5|Chhattisgrah,Raipur

First and second file is separated by '|' and second and third is separated by ','.

After that list is exhausted and reused.

**2. -s (serial):** We can merge the files in sequentially manner using the -s option. It reads all the lines from a single file and merges all these lines into a single line with each line separated by tab. And these single lines are separated by newline.

**$ paste -s number state capital**

1 2 3 4 5

Arunachal Pradesh Assam Andhra Pradesh Bihar Chhattisgrah

Itanagar Dispur Hyderabad Patna Raipur

In the above command, first it reads data from **number** file and merge them into single line with each line separated by tab. After that newline character is introduced and reading from next file i.e. **state** starts and process repeats again till all files are read.

**Combination of -d and -s:** The following example shows how to specify a delimiter for sequential merging of files:

**$ paste -s -d ":" number state capital**

1:2:3:4:5

Arunachal Pradesh:Assam:Andhra Pradesh:Bihar:Chhattisgrah

Itanagar:Dispur:Hyderabad:Patna:Raipur

**3. –version:** This option is used to display the version of paste which is currently running on your system.

$ paste --version

paste (GNU coreutils) 8.26

Packaged by Cygwin (8.26-2)

Copyright (C) 2016 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later .

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Written by David M. Ihnat and David MacKenzie.

**Applications of Paste Command**

**1. Combining N consecutive lines:**The paste command can also be used to merge N consecutive lines from a file into a single line. Here N can be specified by specifying number hyphens(-) after paste.

With 2 hyphens

**$ cat capital | paste - -**

Itanagar Dispur

Hyderabad Patna

Raipur

With 3 hyphens

**$ paste - - - < capital**

Itanagar Dispur Hyderabad

Patna Raipur

**2. Combination with other commands:** Even though **paste** require at least two files for concatenating lines, but data from one file can be given from shell. Like in our example below, **cut** command is used with **-f** option for cutting out first field of **state** file and output is pipelined with paste command having one file name and instead of second file name hyphen is specified.

**Note:** If hyphen is not specified then input from shell is not pasted.

Without hypen

**$ cut -d " " -f 1 state | paste number**

1

2

3

4

5

With hypen

**$ cut -d " " -f 1 state | paste number -**

1 Arunachal

2 Assam

3 Andhra

4 Bihar

5 Chhattisgrah

Ordering of pasting can be changed by altering the location of hyphen:

**$ cut -d " " -f 1 state | paste - number**

Arunachal 1

Assam 2

Andhra 3

Bihar 4

Chhattisgrah 5

diff stands for **difference**. This command is used to display the differences in the files by comparing the files line by line. Unlike its fellow members, [cmp](https://www.geeksforgeeks.org/cmp-command-linux-examples/) and [comm](https://www.geeksforgeeks.org/comm-command-linux-examples/), it tells us which lines in one file have is to be changed to make the two files identical.

The important thing to remember is that **diff**uses certain **special symbols** and **instructions** that are required to make two files identical. It tells you the instructions on how to change the first file to make it match the second file.

**Special symbols are:**

**a : add**

**c :** **change**

**d :** **delete**

**Syntax :**

**diff [options] File1 File2**

Lets say we have two files with names **a.txt** and **b.txt** containing 5 Indian states.

**$ ls**

a.txt b.txt

**$ cat a.txt**

Gujarat

Uttar Pradesh

Kolkata

Bihar

Jammu and Kashmir

**$ cat b.txt**

Tamil Nadu

Gujarat

Andhra Pradesh

Bihar

Uttar pradesh

Now, applying **diff**command without any option we get the following output:

**$ diff a.txt b.txt**

0a1

> Tamil Nadu

2,3c3

< Uttar Pradesh

Andhra Pradesh

5c5

Uttar pradesh

Let’s take a look at what this output means. The first line of the **diff** output will contain:

* Line numbers corresponding to the first file,
* A special symbol and
* Line numbers corresponding to the second file.

Like in our case, **0a1** which means **after**lines 0(at the very beginning of file) you have to add **Tamil Nadu** to match the second file line number 1. It then tells us what those lines are in each file preceeded by the symbol:

* ;Lines preceded by a **<** are lines from the first file.
* Lines preceded by **>** are lines from the second file.
* Next line contains **2,3c3** which means from line 2 to line 3 in the first file needs to be changed to match line number 3 in the second file. It then tells us those lines with the above symbols.
* The three dashes **(“—“)** merely separate the lines of file 1 and file 2.

As a summary to make both the files identical, first add *Tamil Nadu* in the first file at very beginning to match line 1 of second file after that change line 2 and 3 of first file i.e. *Uttar Pradesh* and *Kolkata* with line 3 of second file i.e. *Andhra Pradesh*. After that change line 5 of first file i.e. *Jammu and Kashmir* with line 5 of second file i.e. *Uttar pradesh*.

Now let’s see what it looks like when **diff** tells us that we need to delete a line.

**$ cat a.txt**

Gujarat

Andhra Pradesh

Telangana

Bihar

Uttar pradesh

**$ cat b.txt**

Gujarat

Andhra Pradesh

Bihar

Uttar pradesh

**$ diff a.txt b.txt**

3d2

< Telangana

Here above output **3d2** means delete line 3rd of first file i.e. *Telangana* so that both the files **sync up** at line 2.

**Options**

Linux system offers two different ways to view the **diff** command output i.e. **context mode**and **unified mode**.

1. **-c (context) :** To view differences in context mode, use the **-c** option. Lets try to understand this with example, we have two files **file1.txt** and **file2.txt**:
2. **$ cat file1.txt**
3. cat
4. mv
5. comm
6. cp
7. **$ cat file2.txt**
8. cat
9. cp
10. diff
11. comm
13. **$ diff -c file1.txt file2.txt**
14. \*\*\* file1.txt Thu Jan 11 08:52:37 2018
15. --- file2.txt Thu Jan 11 08:53:01 2018
16. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
17. \*\*\* 1,4 \*\*\*\*
18. cat
19. - mv
20. - comm
21. cp
22. --- 1,4 ----
23. cat
24. cp
25. + diff
26. + comm

The first file is indicated by **\*\*\***, and the second file is indicated by **—**.  
The line with **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** is just a separator.

The first two lines of this output show us information about **file 1** and **file 2**. It lists the file name, modification date, and modification time of each of our files, one per line.  
The next line has three asterisks **\*\*\*** followed by a line range from the first file (in our case lines 1 through 4, separated by a comma). Then four asterisks **\*\*\*\***. After that it shows the contents of the first file with the following indicators:

**(i)** If the line needs to be unchanged, it is prefixed by two spaces.  
**(ii)** If the line needs to be changed, it is prefixed by an symbol and a space. The symbol means are as follows:

**(a) + :** It indicates a line in the second file that needs to be added to the first file to make them identical.  
**(b) – :** It indicates a line in the first file that needs to be deleted to make them identical.  
Like in our case, it is needed to delete *mv* and *comm* from first file and add *diff* and *comm* to the first file to make both of them identical.

After that there are three dashes **—** followed by a line range from the second file (in our case lines 1 through 4, separated by a comma). Then four dashes **—-**. Then it shows the contents of the second file.

1. **-u (unified) :** To view differences in unified mode, use the **-u** option. It is similar to context mode but it **doesn’t display any redundant information** or it shows the information in concise form.
2. **$ cat file1.txt**
3. cat
4. mv
5. comm
6. cp
7. **$ cat file2.txt**
8. cat
9. cp
10. diff
11. comm
12. **$ diff -u file1.txt file2.txt**
13. --- file1.txt 2018-01-11 10:39:38.237464052 +0000
14. +++ file2.txt 2018-01-11 10:40:00.323423021 +0000
15. @@ -1,4 +1,4 @@
16. cat
17. -mv
18. -comm
19. cp
20. +diff
21. +comm

The first file is indicated by **—**, and the second file is indicated by **+++**.  
The first two lines of this output show us information about **file 1** and **file 2**. It lists the file name, modification date, and modification time of each of our files, one per line.  
After that the next line has two at sign **@** followed by a line range from the first file (in our case lines 1 through 4, separated by a comma) prefixed by **–** and then space and then again followed by a line range from the second file prefixed by **+** and at the end two at sign **@**. Followed by the file content in output tells us which line remain unchanged and which lines needs to added or deleted(indicated by symbols) in the **file 1** to make it identical to **file 2**.

1. **-i :** By default this command is *case sensitive*. To make this command *case in-sensitive* use **-i** option with **diff**.
2. **$ cat file1.txt**
3. dog
4. mv
5. CP
6. comm
7. **$ cat file2.txt**
8. DOG
9. cp
10. diff
11. comm
12. Without using this option:
13. **$ diff file1.txt file2.txt**
14. 1,3c1,3
15. < dog
16. < mv
17. DOG
18. > cp
19. > diff
20. Using this option:
21. **$ diff -i file1.txt file2.txt**
22. 2d1
23. diff
24. **–version :** This option is used to display the version of **diff**which is currently running on your system.
25. $ diff --version
26. diff (GNU diffutils) 3.5
27. Packaged by Cygwin (3.5-2)
28. Copyright (C) 2016 Free Software Foundation, Inc.
29. License GPLv3+: GNU GPL version 3 or later .
30. This is free software: you are free to change and redistribute it.
31. There is NO WARRANTY, to the extent permitted by law.
32. Written by Paul Eggert, Mike Haertel, David Hayes,
33. Richard Stallman, and Len Tower.

The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by **byte position, character and field**. Basically the cut command slices a line and extracts the text. It is necessary to specify option with command otherwise it gives error. If more than one file name is provided then data from each file is **not precedes** by its file name.

**Syntax:**

**cut OPTION... [FILE]...**

Let us consider two files having name **state.txt** and **capital.txt** contains 5 names of the Indian states and capitals respectively.

$ cat state.txt

Andhra Pradesh

Arunachal Pradesh

Assam

Bihar

Chhattisgarh

Without any option specified it displays error.

**$ cut state.txt**

cut: you must specify a list of bytes, characters, or fields

Try 'cut --help' for more information.

**Options and their Description with examples:**

**1. -b(byte):**To extract the specific bytes, you need to follow -b option with the list of byte numbers separated by comma. Range of bytes can also be specified using the hyphen(-). It is necessary to specify list of byte numbers otherwise it gives error. **Tabs and backspaces** are treated like as a character of 1 byte.

**List without ranges**

**$ cut -b 1,2,3 state.txt**

And

Aru

Ass

Bih

Chh

**List with ranges**

**$ cut -b 1-3,5-7 state.txt**

Andra

Aruach

Assm

Bihr

Chhtti

It uses a special form for selecting bytes from beginning upto the end of the line:

In this, 1- indicate from 1st byte to end byte of a line

**$ cut -b 1- state.txt**

Andhra Pradesh

Arunachal Pradesh

Assam

Bihar

Chhattisgarh

In this, -3 indicate from 1st byte to 3rd byte of a line

**$ cut -b -3 state.txt**

And

Aru

Ass

Bih

Chh

**2. -c (column):** To cut by character use the -c option. This selects the characters given to the -c option. This can be a list of numbers separated comma or a range of numbers separated by hyphen(-). **Tabs and backspaces** are treated as a character. It is necessary to specify list of character numbers otherwise it gives error with this option.

**Syntax:**

**$cut -c [(k)-(n)/(k),(n)/(n)] filename**

Here,**k** denotes the starting position of the character and **n** denotes the ending position of the character in each line, if *k* and *n* are separated by “-” otherwise they are only the position of character in each line from the file taken as an input.

**$ cut -c 2,5,7 state.txt**

nr

rah

sm

ir

hti

Above cut command prints second, fifth and seventh character from each line of the file.

**$ cut -c 1-7 state.txt**

Andhra

Arunach

Assam

Bihar

Chhatti

Above cut command prints first seven characters of each line from the file.

Cut uses a special form for selecting characters from beginning upto the end of the line:

**$ cut -c 1- state.txt**

Andhra Pradesh

Arunachal Pradesh

Assam

Bihar

Chhattisgarh

Above command prints starting from first character to end. Here in command only starting

position is specified and the ending position is omitted.

**$ cut -c -5 state.txt**

Andhr

Aruna

Assam

Bihar

Chhat

Above command prints starting position to the fifth character. Here the starting position

is omitted and the ending position is specified.

**3. -f (field):** **-c** option is useful for fixed-length lines. Most unix files doesn’t have fixed-length lines. To extract the useful information you need to cut by fields rather than columns. List of the fields number specified must be separated by comma. *Ranges are not described with -f option*. **cut**uses **tab**as a default field delimiter but can also work with other delimiter by using **-d** option.  
**Note:** Space is not considered as delimiter in UNIX.

**Syntax:**

**$cut -d "delimiter" -f (field number) file.txt**

Like in the file **state.txt** fields are separated by space if -d option is not used then it prints whole line:

**$ cut -f 1 state.txt**

Andhra Pradesh

Arunachal Pradesh

Assam

Bihar

Chhattisgarh

If -d option is used then it considered space as a field separator or delimiter:

**$ cut -d " " -f 1 state.txt**

Andhra

Arunachal

Assam

Bihar

Chhattisgarh

Command prints field from first to fourth of each line from the file.

**Command:**

$ cut -d " " -f 1-4 state.txt

**Output:**

Andhra Pradesh

Arunachal Pradesh

Assam

Bihar

Chhattisgarh

**4. –complement:** As the name suggests it complement the output. This option can be used in the combination with other options either with **-f** or with **-c**.

**$ cut --complement -d " " -f 1 state.txt**

Pradesh

Pradesh

Assam

Bihar

Chhattisgarh

**$ cut --complement -c 5 state.txt**

Andha Pradesh

Arunchal Pradesh

Assa

Biha

Chhatisgarh

**5. –output-delimiter:** By default the output delimiter is same as input delimiter that we specify in the cut with **-d** option. To change the output delimiter use the option **–output-delimiter=”delimiter”**.

**$ cut -d " " -f 1,2 state.txt --output-delimiter='%'**

Andhra%Pradesh

Arunachal%Pradesh

Assam

Bihar

Chhattisgarh

Here cut command changes delimiter(%) in the standard output between the fields which is specified by using -f option .

**6. –version:** This option is used to display the version of cut which is currently running on your system.

**$ cut --version**

cut (GNU coreutils) 8.26

Packaged by Cygwin (8.26-2)

Copyright (C) 2016 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later .

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Written by David M. Ihnat, David MacKenzie, and Jim Meyering.

**Applications of cut Command**

**1. How to use tail with pipes(|):** The cut command can be piped with many other commands of the unix. In the following example output of the **cat**command is given as input to the **cut**command with **-f** option to sort the state names coming from file state.txt in the reverse order.

**$ cat state.txt | cut -d ' ' -f 1 | sort -r**

Chhattisgarh

Bihar

Assam

Arunachal

Andhra

It can also be piped with one or more filters for additional processing. Like in the following example, we are using cat, head and cut command and whose output is stored in the file name list.txt using directive(>).

**$ cat state.txt | head -n 3 | cut -d ' ' -f 1 > list.txt**

**$ cat list.txt**

Andhra

Arunachal

Assam

uniq Command in LINUX with examples

The **uniq** command in Linux is a command line utility that reports or filters out the repeated lines in a file.  
In simple words, **uniq** is the tool that helps to detect the adjacent duplicate lines and also deletes the duplicate lines. **uniq** filters out the adjacent matching lines from the input file(that is required as an argument) and writes the filtered data to the output file .

**Syntax of uniq Command :**

//...syntax of uniq...//

**$uniq [OPTION] [INPUT[OUTPUT]]**

The syntax of this is quite easy to understand. Here, **INPUT** refers to the input file in which repeated lines need to be filtered out and if INPUT isn’t specified then **uniq** reads from the standard input. **OUTPUT** refers to the output file in which you can store the filtered output generated by **uniq** command and as in case of INPUT if OUTPUT isn’t specified then **uniq**writes to the standard output.

Now, let’s understand the use of this with the help of an example. Suppose you have a text file named kt.txt which contains repeated lines that needs to be omitted. This can simply be done with uniq.

//displaying contents of kt.txt//

**$cat kt.txt**

I love music.

I love music.

I love music.

I love music of Kartik.

I love music of Kartik.

Thanks.

Now, as we can see that the above file contains multiple duplicate lines. Now, lets’s use uniq command to remove them:

//...using uniq command.../

**$uniq kt.txt**

I love music.

I love music of Kartik.

Thanks.

/\* with the use of uniq all

the repeated lines are removed\*/

As you can see that we just used the name of input file in the above uniq example and as we didn’t use any output file to store the produced output, the uniq command displayed the filtered output on the standard output with all the duplicate lines removed.

**Note: uniq** isn’t able to detect the duplicate lines unless they are adjacent. The content in the file must be therefore sorted before using uniq or you can simply use **sort -u** instead f uniq.

**Options For uniq Command:**

1. **-c – -count :** It tells how many times a line was repeated by displaying a number as a prefix with the line.
2. **-d – -repeated :** It only prints the repeated lines and not the lines which aren’t repeated.
3. **-D – -all-repeated[=METHOD] :** It prints all duplicate lines and METHOD can be any of the following:
   * **none :** Do not delimit duplicate lines at all. This is the default.
   * **prepend :** Insert a blank line before each set of duplicated lines.
   * **separate :** Insert a blank line between each set of duplicated lines.
4. **-f N – -skip-fields(N) :** It allows you to skip N fields(a field is a group of characters, delimited by whitespace) of a line before determining uniqueness of a line.
5. **-i – -ignore case :** By default, comparisons done are case sensitive but with this option case insensitive comparisons can be made.
6. **-s N – -skip-chars(N) :** It doesn’t compares the first N characters of each line while determining uniqueness. This is like the -f option, but it skips individual characters rather than fields.
7. **-u – -unique :** It allows you to print only unique lines.
8. **-z – -zero-terminated :** It will make a line end with 0 byte(NULL), instead of a newline.
9. **-w N – -check-chars(N) :** It only compares N characters in a line.
10. **– – help :** It displays a help message and exit.
11. **– – version :** It displays version information and exit.

**Examples of uniq with Options**

**1. Using -c option :** It tells the number of times a line was repeated.

//using uniq with -c//

**$uniq -c kt.txt**

3 I love music.

1

2 I love music of Kartik.

1

1 Thanks.

/\*at the starting of each

line its repeated number is

displayed\*/

**2. Using -d option :** It only prints the repeated lines.

//using uniq with -d//

**$uniq -d kt.txt**

I love music.

I love music of Kartik.

/\*it only displayed one

duplicate line per group\*/

**3. Using -D option :** It also prints only duplicate lines but not one per group.

//using -D option//

**$uniq -D kt.txt**

I love music.

I love music.

I love music.

I love music of Kartik.

I love music of Kartik.

/\* all the duplicate lines

are displayed\*/

**4. Using -u option :** It prints only the unique lines.

//using -u option//

**$uniq -u kt.txt**

Thanks.

/\*only unique lines are

displayed\*/

**5. Using -f N option :** As told above, this allows the N fields to be skipped while comparing uniqueness of the lines. This option is helpful when the lines are numbered as shown in the example below:

//displaying contents of f1.txt//

**$cat f1.txt**

1. I love music.

2. I love music.

3. I love music of Kartik.

4. I love music of Kartik.

//now using uniq with -f N option//

**$uniq -f 2 f1.txt**

1. I love music.

3. I love music of Kartik.

/\*2 is used cause we needed to

compare the lines after the

numbering 1,2.. and after dots\*/

**6. Using -s N option :**This is similar to -f N option but it skips N characters but not N fields.

//displaying content of f2.txt//

**$cat f2.txt**

#%@I love music.

^&(I love music.

\*-!@thanks.

#%@!thanks.

//now using -s N option//

**$uniq -s 3 f2.txt**

#%@I love music.

\*-!@thanks.

#%@!thanks.

/\*lines same after skipping

3 characters are filtered\*/

**7. Using -w option :** Similar to the way of skipping characters, we can also ask uniq to limit the comparison to a set number of characters. For this, -w command line option is used.

//displaying content of f3.txt//

**$cat f3.txt**

How it is possible?

How it can be done?

How to use it?

//now using -w option//

**$uniq -w 3 f3.txt**

How

/\*as the first 3 characters

of all the 3 lines are same

that's why uniq treated all these

as duplicates and gave output

accordingly\*/

**8. Using -i option :** It is used to make the comparison case-insensitive.

//displaying contents of f4.txt//

**$cat f4.txt**

I LOVE MUSIC

i love music

THANKS

//using uniq command//

**$uniq f4.txt**

I LOVE MUSIC

i love music

THANKS

/\*the lines aren't treated

as duplicates with simple

use of uniq\*/

//now using -i option//

**$uniq -i f4.txt**

I LOVE MUSIC

THANKS

/\*now second line is removed

when -i option is used\*/

**9. Using -z option :** By default, the output uniq produces is newline terminated. However, if you want, you want to have a NULL terminated output instead (useful while dealing with uniq in scripts). This can be made possible using the -z command line option.

Syntax:

//syntax of using uniq

with -z option//

**$uniq -z file-name**