Linux and Unix cut command

About cut

Remove or "cut out" sections of each line of a file or files.

Syntax

cut OPTION... [FILE]...

Options-b, --bytes=LIST Select only the bytes from each line as specified in LIST. LIST specifies a byte, a set of bytes, or a range of bytes; see Specifying LIST below.

-c, --characters=LIST Select only the characters from each line as specified in LIST. LIST specifies a character, a set of characters, or a range of characters; see Specifying LIST below.

-d, --delimiter=DELIM use character DELIM instead of a tab for the field delimiter.

-f, --fields=LIST select only these fields on each line; also print any line that contains no delimiter character, unless the -s option is specified. LIST specifies a field, a set of fields, or a range of fields; see Specifying LIST below.

-n This option is ignored, but is included for compatibility reasons.

--complement complement the set of selected bytes, characters or fields.

-s, --only-delimited do not print lines not containing delimiters.

--output-delimiter=STRING use STRING as the output delimiter string. The default is to use the input delimiter.

--help Display a help message and exit.

--version output version information and exit.

Usage Notes

When invoking cut, use the -b, -c, or -f option, but only one of them.

If no FILE is specified, cut reads from the standard input.

Specifying LIST

Each LIST is made up of an integer, a range of integers, or multiple integer ranges separated by commas. Selected input is written in the same order that it is read, and is written to output exactly once. A range consists of:N the Nth byte, character, or field, counted from 1.

N- from the Nth byte, character, or field, to the end of the line.

N-M from the Nth to the Mth byte, character, or field (inclusive).

-M from the first to the Mth byte, character, or field.

For example, let's say you have a file named data.txt which contains the following text:

one two three four five

alpha beta gamma delta epsilon

In this example, each of these words is separated by a tab character, not spaces. The tab character is the default delimiter of cut, so it will by default consider a field to be anything delimited by a tab.

To "cut" only the third field of each line, use the command:

cut -f 3 data.txt

...which will output the following:

three

gamma

If instead you want to "cut" only the second-through-fourth field of each line, use the command:

cut -f 2-4 data.txt

...which will output the following:

two three four

beta gamma delta

If you want to "cut" only the first-through-second and fourth-through-fifth field of each line (omitting the third field), use the command:

cut -f 1-2,4-5 data.txt

...which will output the following:

one two four five

alpha beta delta epsilon

Or, let's say you want the third field and every field after it, omitting the first two fields. In this case, you could use the command:

cut -f 3- data.txt

...which will output the following:

three four five

gamma delta epsilon

Specifying a range with LIST also applies to cutting characters (-c) or bytes (-b) from a line. For example, to output only the third-through-twelfth character of every line of data.txt, use the command:

cut -c 3-12 data.txt

...which will output the following:

e two thre

pha beta g

Remember that the "space" in between each word is actually a single tab character, so both lines of output are displaying ten characters: eight alphanumeric characters and two tab characters. In other words, cut is omitting the first two characters of each line, counting tabs as one character each; outputting characters three through twelve, counting tabs as one character each; and omitting any characters after the twelfth.

Counting bytes instead of characters will result in the same output in this case, because in an ASCII-encoded text file, each character is represented by a single byte (eight bits) of data. So the command:

cut -b 3-12 data.txt

...will, for our file data.txt, produce exactly the same output:

e two thre

pha beta g

Specifying A Delimiter Other Than Tab

The tab character is the default delimiter that cut uses to determine what constitutes a field. So, if your file's fields are already delimited by tabs, you don't need to specify a different delimiter character.

You can specify any character as the delimiter, however. For instance, the file /etc/passwd contains information about each user on the system, one user per line, and each information field is delimited by a colon (":"). For example, the line of /etc/passwd for the root user may look like this:

root:x:0:0:root:/root:/bin/bash

These fields contain the following information, in the following order, separated by a colon character:

Username

Password (shown as x if encrypted)

User ID number (UID)

Group ID number (GID)

Comment field (used by the finger command)

Home Directory

Shell

The username is the first field on the line, so to display each username on the system, use the command:

cut -f 1 -d ':' /etc/passwd

...which will output, for example:

root

daemon

bin

sys

chope

(There are many more user accounts on a typical system, including many accounts specific to system services, but for this example we will pretend there are only five users.)

The third field of each line in the /etc/passwd file is the UID (user ID number), so to display each username and user ID number, use the command:

cut -f 1,3 -d ':' /etc/passwd

...which will output the following, for example:

root:0

daemon:1

bin:2

sys:3

chope:1000

As you can see, the output will be delimited, by default, using the same delimiter character specified for the input. In this case, that's the colon character (":"). You can specify a different delimiter for the input and output, however. So, if you wanted to run the previous command, but have the output delimited by a space, you could use the command:

cut -f 1,3 -d ':' --output-delimiter=' ' /etc/passwd

root 0

daemon 1

bin 2

sys 3

chope 1000

But what if you want the output to be delimited by a tab? Specifying a tab character on the command line is a bit more complicated, because it is an unprintable character. To specify it on the command line, you must "protect" it from the shell. This is done differently depending on which shell you're using, but in the Linux default shell (bash), you can specify the tab character with $'\t'. So the command:

cut -f 1,3 -d ':' --output-delimiter=$'\t' /etc/passwd

...will output the following, for example:

root 0

daemon 1

bin 2

sys 3

chope 1000

Examples

cut -c 3 file.txt

Outputs the third character of every line of the file file.txt, omitting the others.

cut -c 1-3 file.txt

Outputs the first three characters of every line of the file file.txt, omitting the rest.

cut -c 3- file.txt

Outputs the third through the last characters of each line of the file file.txt, omitting the first two characters.

cut -d ':' -f 1 /etc/passwd

Outputs the first field of the file /etc/passwd, where fields are delimited by a colon (':'). The first field of /etc/passwd is the username, so this command will output every username in the passwd file.

grep '/bin/bash' /etc/passwd | cut -d ':' -f 1,6

Outputs the first and sixth fields, delimited by a colon, of any entry in the /etc/passwd file which specifies /bin/bash as the login shell. This command will output the username and home directory of any user whose login shell is /bin/bash.

Linux and Unix paste command

About paste

The paste command displays the corresponding lines of multiple files side-by-side.

Syntax

paste [OPTION]... [FILE]...

Description

paste writes lines consisting of the sequentially corresponding lines from each FILE, separated by tabs, to the standard output. With no FILE, or when FILE is a dash ("-"), paste reads from standard input.

Options-d, --delimiters=LIST reuse characters from LIST instead of tabs.

-s, --serial paste one file at a time instead of in parallel.

--help Display a help message, and exit.

--version Display version information, and exit.

Examples

paste file1.txt file2.txt

This command would display the contents of file1.txt and file2.txt, side-by-side, with the corresponding lines of each file separated by a tab.

Paste command is one of the useful commands in unix or linux operating system. The paste command merges the lines from multiple files. The paste command sequentially writes the corresponding lines from each file separated by a TAB delimiter on the unix terminal.

The syntax of the paste command is

paste [options] files-list

The options of paste command are:

-d : Specify of a list of delimiters.

-s : Paste one file at a time instead of in parallel.

--version : version information

--help : Help about the paste command.

Paste Command Examples:

Create the following three files in your unix or linux servers to practice to practice the examples:

> cat file1

Unix

Linux

Windows

> cat file2

Dedicated server

Virtual server

> cat file3

Hosting

Machine

Operating system

1. Merging files in parallel

By default, the paste command merges the files in parallel. The paste command writes corresponding lines from the files as a tab delimited on the terminal.

> paste file1 file2

Unix Dedicated server

Linux Virtual server

Windows

> paste file2 file1

Dedicated server Unix

Virtual server Linux

Windows

2. Specifying the delimiter

Paste command uses the tab delimiter by default for merging the files. You can change the delimiter to any other character by using the -d option.

> paste -d"|" file1 file2

Unix|Dedicated server

Linux|Virtual server

Windows|

In the above example, pipe delimiter is specified

3. Merging files in sequentially.

You can merge the files in sequentially using the -s option. The paste command reads each file in sequentially. It reads all the lines from a single file and merges all these lines into a single line.

> paste -s file1 file2

Unix Linux Windows

Dedicated server Virtual server

The following example shows how to specify a delimiter for sequential merging of files:

> paste -s -d"," file1 file2

Unix,Linux,Windows

Dedicated server,Virtual server

4. Specifying multiple delimiters.

Multiple delimiters come in handy when you want to merge more than two files with different delimiters. For example I want to merge file1, file2 with pipe delimiter and file2, file3 with comma delimiter. In this case multiple delimiters will be helpful.

> paste -d"|," file1 file2 file3

Unix|Dedicated server,Hosting

Linux|Virtual server,Machine

Windows|,Operating system

5. Combining N consecutive lines

The paste command can also be used to merge N consecutive lines from a file into a single line. The following example merges 2 consecutive lines into a single line

> cat file1 | paste - -

Unix Linux

Windows

**Grep Command Examples In Linux / UNIX**

First create the following demo\_file that will be used in the examples below to demonstrate grep command.

$ cat demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

This Line Has All Its First Character Of The Word With Upper Case.

Two lines above this line is empty.

And this is the last line.

1. Search for the given string in a single file

The basic usage of grep command is to search for a specific string in the specified file as shown below.

Syntax:

grep "literal\_string" filename

$ grep "this" demo\_file

this line is the 1st lower case line in this file.

Two lines above this line is empty.

And this is the last line.

2. Checking for the given string in multiple files.

Syntax:

grep "string" FILE\_PATTERN

This is also a basic usage of grep command. For this example, let us copy the demo\_file to demo\_file1. The grep output will also include the file name in front of the line that matched the specific pattern as shown below. When the Linux shell sees the meta character, it does the expansion and gives all the files as input to grep.

$ cp demo\_file demo\_file1

$ grep "this" demo\_\*

demo\_file:this line is the 1st lower case line in this file.

demo\_file:Two lines above this line is empty.

demo\_file:And this is the last line.

demo\_file1:this line is the 1st lower case line in this file.

demo\_file1:Two lines above this line is empty.

demo\_file1:And this is the last line.

3. Case insensitive search using grep -i

Syntax:

grep -i "string" FILE

This is also a basic usage of the grep. This searches for the given string/pattern case insensitively. So it matches all the words such as “the”, “THE” and “The” case insensitively as shown below.

$ grep -i "the" demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

This Line Has All Its First Character Of The Word With Upper Case.

And this is the last line.

4. Match regular expression in files

Syntax:

grep "REGEX" filename

This is a very powerful feature, if you can use use regular expression effectively. In the following example, it searches for all the pattern that starts with “lines” and ends with “empty” with anything in-between. i.e To search “lines[anything in-between]empty” in the demo\_file.

$ grep "lines.\*empty" demo\_file

Two lines above this line is empty.

From documentation of grep: A regular expression may be followed by one of several repetition operators:

? The preceding item is optional and matched at most once.

\* The preceding item will be matched zero or more times.

+ The preceding item will be matched one or more times.

{n} The preceding item is matched exactly n times.

{n,} The preceding item is matched n or more times.

{,m} The preceding item is matched at most m times.

{n,m} The preceding item is matched at least n times, but not more than m times.

5. Checking for full words, not for sub-strings using grep -w

If you want to search for a word, and to avoid it to match the substrings use -w option. Just doing out a normal search will show out all the lines.

The following example is the regular grep where it is searching for “is”. When you search for “is”, without any option it will show out “is”, “his”, “this” and everything which has the substring “is”.

$ grep -i "is" demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

This Line Has All Its First Character Of The Word With Upper Case.

Two lines above this line is empty.

And this is the last line.

The following example is the WORD grep where it is searching only for the word “is”. Please note that this output does not contain the line “This Line Has All Its First Character Of The Word With Upper Case”, even though “is” is there in the “This”, as the following is looking only for the word “is” and not for “this”.

$ grep -iw "is" demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

Two lines above this line is empty.

And this is the last line.

6. Displaying lines before/after/around the match using grep -A, -B and -C

When doing a grep on a huge file, it may be useful to see some lines after the match. You might feel handy if grep can show you not only the matching lines but also the lines after/before/around the match.

Please create the following demo\_text file for this example.

$ cat demo\_text

4. Vim Word Navigation

You may want to do several navigation in relation to the words, such as:

\* e - go to the end of the current word.

\* E - go to the end of the current WORD.

\* b - go to the previous (before) word.

\* B - go to the previous (before) WORD.

\* w - go to the next word.

\* W - go to the next WORD.

WORD - WORD consists of a sequence of non-blank characters, separated with white space.

word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

\* 192.168.1.1 - seven words.

6.1 Display N lines after match

-A is the option which prints the specified N lines after the match as shown below.

Syntax:

grep -A <N> "string" FILENAME

The following example prints the matched line, along with the 3 lines after it.

$ grep -A 3 -i "example" demo\_text

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

\* 192.168.1.1 - seven words.

6.2 Display N lines before match

-B is the option which prints the specified N lines before the match.

Syntax:

grep -B <N> "string" FILENAME

When you had option to show the N lines after match, you have the -B option for the opposite.

$ grep -B 2 "single WORD" demo\_text

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

6.3 Display N lines around match

-C is the option which prints the specified N lines before the match. In some occasion you might want the match to be appeared with the lines from both the side. This options shows N lines in both the side(before & after) of match.

$ grep -C 2 "Example" demo\_text

word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

7. Highlighting the search using GREP\_OPTIONS

As grep prints out lines from the file by the pattern / string you had given, if you wanted it to highlight which part matches the line, then you need to follow the following way.

When you do the following export you will get the highlighting of the matched searches. In the following example, it will highlight all the this when you set the GREP\_OPTIONS environment variable as shown below.

$ export GREP\_OPTIONS='--color=auto' GREP\_COLOR='100;8'

$ grep this demo\_file

this line is the 1st lower case line in this file.

Two lines above this line is empty.

And this is the last line.

8. Searching in all files recursively using grep -r

When you want to search in all the files under the current directory and its sub directory. -r option is the one which you need to use. The following example will look for the string “ramesh” in all the files in the current directory and all it’s subdirectory.

$ grep -r "ramesh" \*

9. Invert match using grep -v

You had different options to show the lines matched, to show the lines before match, and to show the lines after match, and to highlight match. So definitely You’d also want the option -v to do invert match.

When you want to display the lines which does not matches the given string/pattern, use the option -v as shown below. This example will display all the lines that did not match the word “go”.

$ grep -v "go" demo\_text

4. Vim Word Navigation

You may want to do several navigation in relation to the words, such as:

WORD - WORD consists of a sequence of non-blank characters, separated with white space.

word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

\* 192.168.1.1 - seven words.

10. display the lines which does not matches all the given pattern.

Syntax:

grep -v -e "pattern" -e "pattern"

$ cat test-file.txt

a

b

c

d

$ grep -v -e "a" -e "b" -e "c" test-file.txt

d

11. Counting the number of matches using grep -c

When you want to count that how many lines matches the given pattern/string, then use the option -c.

Syntax:

grep -c "pattern" filename

$ grep -c "go" demo\_text

6

When you want do find out how many lines matches the pattern

$ grep -c this demo\_file

3

When you want do find out how many lines that does not match the pattern

$ grep -v -c this demo\_file

4

12. Display only the file names which matches the given pattern using grep -l

If you want the grep to show out only the file names which matched the given pattern, use the -l (lower-case L) option.

When you give multiple files to the grep as input, it displays the names of file which contains the text that matches the pattern, will be very handy when you try to find some notes in your whole directory structure.

$ grep -l this demo\_\*

demo\_file

demo\_file1

13. Show only the matched string

By default grep will show the line which matches the given pattern/string, but if you want the grep to show out only the matched string of the pattern then use the -o option.

It might not be that much useful when you give the string straight forward. But it becomes very useful when you give a regex pattern and trying to see what it matches as

$ grep -o "is.\*line" demo\_file

is line is the 1st lower case line

is line

is is the last line

14. Show the position of match in the line

When you want grep to show the position where it matches the pattern in the file, use the following options as

Syntax:

grep -o -b "pattern" file

$ cat temp-file.txt

12345

12345

$ grep -o -b "3" temp-file.txt

2:3

8:3

Note: The output of the grep command above is not the position in the line, it is byte offset of the whole file.

15. Show line number while displaying the output using grep -n

To show the line number of file with the line matched. It does 1-based line numbering for each file. Use -n option to utilize this feature.

$ grep -n "go" demo\_text

5: \* e - go to the end of the current word.

6: \* E - go to the end of the current WORD.

7: \* b - go to the previous (before) word.

8: \* B - go to the previous (before) WORD.

9: \* w - go to the next word.

10: \* W - go to the next WORD.