**read** is a [builtin](https://www.computerhope.com/jargon/b/builtin.htm) command of the [Bash](https://www.computerhope.com/unix/ubash.htm) shell, which reads a line of text from [standard input](https://www.computerhope.com/jargon/s/stdin.htm) and splits it into words. These words can then be used as the input for other commands.

Syntax

**read** [**-ers**] [**-a** *array*] [**-d** *delim*] [**-i** *text*] [**-n** *nchars*] [**-N** *nchars*]

[**-p** *prompt*] [**-t** *timeout*] [**-u** *fd*] [*name* ...] [*name2* ...]

**Description**

**read** reads a single line from standard input, or from the file descriptor *fd* if the **-u**option is used (see **-u**, below).

By default, **read** considers a [newline](https://www.computerhope.com/jargon/n/newline.htm) character as the end of a line, but this can be changed using the **-d** option.

After reading, the line is split into words according to the value of the special shell variable **IFS**, the internal field separator. By default, the IFS value is "space, tab, or newline". If **read** encounters readable [characters](https://www.computerhope.com/jargon/c/charact.htm) before encountering an IFS, it considers those characters to be a word. (For more information about the IFS, see [word splitting in bash](https://www.computerhope.com/unix/ubash.htm#word-splitting).)

**Tip:** To preserve [white space](https://www.computerhope.com/jargon/w/whitspac.htm) at the beginning or the end of a line, it's common to specify **IFS=** (with no value) immediately before the **read** command. After reading is completed, the IFS returns to its previous value. For more about this, see the [examples](https://www.computerhope.com/unix/bash/read.htm#examples) below.

**read** assigns the first word it finds to *name*, the second word to *name2*, etc. If there are more words than names, all remaining words are assigned to the last *name* specified. If only a single *name* is specified, the entire line is assigned to that variable.

If no *name* is specified, the input is stored in a variable named **REPLY**.

**Options**

The **read** builtin command takes the following options:

|  |  |
| --- | --- |
| **-a** *array* | Store the words in an indexed [array](https://www.computerhope.com/jargon/a/array.htm) named *array*. Numbering of array elements starts at zero. |
| **-d** *delim* | Set the [delimiter](https://www.computerhope.com/jargon/d/delimite.htm) character to *delim*. This character will signal the end of the line. If **-d** is not used, the default line delimiter is a [newline](https://www.computerhope.com/jargon/n/newline.htm). |
| **-e** | Get a line of input from an interactive shell. The user manually inputs characters until the line delimiter is reached. |
| **-i** *text* | When used in conjunction with **-e** (and only if **-s** is not used), *text* is inserted as the initial text of the input line. The user is permitted to edit *text* on the input line. |
| **-n** *nchars* | Stop reading after an [integer](https://www.computerhope.com/jargon/i/integer.htm) number *nchars* characters have been read, if the line delimiter has not been reached. |
| **-N** *nchars* | Ignore the line delimiter. Stop reading only after *nchars* characters have been read, [EOF](https://www.computerhope.com/jargon/e/eof.htm) is reached, or read times out (see **-t**). |
| **-p** *prompt* | Print the string *prompt*, without a newline, before beginning to read. |
| **-r** | Use "raw input". Specifically, this option causes **read** to interpret backslashes [literally](https://www.computerhope.com/jargon/l/literal.htm), rather than interpreting them as [escape characters](https://www.computerhope.com/jargon/e/esc.htm#escape-char). |
| **-s** | Do not echo keystrokes when **read** is taking input from the [terminal](https://www.computerhope.com/jargon/t/terminal.htm). |
| **-t** *timeout* | [Time out](https://www.computerhope.com/jargon/t/timeout.htm), and return failure, if a complete line of input is not read within *timeout*seconds. If the timeout value is zero, **read** will not read any data, but will return success if input was available to read. If *timeout* is not specified, the value of the shell variable **TMOUT** is used instead, if it exists. The value of *timeout* can be a fractional number, e.g., **3.5**. |
| **-u** *fd* | Read from the file descriptor *fd* instead of standard input. The file descriptor should be a small integer. For information about opening a custom file descriptor, see [opening file descriptors in bash](https://www.computerhope.com/unix/ubash.htm#opening-file-descriptors). |

**Exit status**

The exit status of **read** is zero unless EOF is encountered, the timeout is exceeded, an error occurs assigning a value to *name*, or the file descriptor provided to **-u** is invalid.

If a timeout is exceeded, the exit status will be a number greater than 128.

Examples

while read; do echo "$REPLY"; done

**read** will take data from the terminal. Type whatever you'd like, and press Enter. The text is [echoed](https://www.computerhope.com/unix/uecho.htm) on the next line. This loop will continue until you press **Ctrl**+**D** (EOF) on a new line. Because no variable names were specified, the entire line of text is stored in the variable **REPLY**.

while read text; do echo "$text"; done

Same as above, using the variable name **text**.

while read -ep "Type something: " -i "My text is " text; do

echo "$text";

done

Provides a [prompt](https://www.computerhope.com/jargon/p/prompt.htm), and initial text for user input. The user can erase "My text is ", or leave it as part of the input. Typing **Ctrl**+**D** on a new line terminates the loop.

echo "Hello, world!" | (read; echo "$REPLY")

Enclosing the **read** and **echo** commands in parentheses executes them in a dedicated subshell. This allows the **REPLY** variable to be accessed by both **read** and **echo**. For more information, see [bash command execution environments: subshells](https://www.computerhope.com/unix/ubash.htm#command-execution-environment-subshell).

echo "one two three four" | while read word1 word2 word3; do

echo "word1: $word1"

echo "word2: $word2"

echo "word3: $word3"

done

Echo "one two three four" and [pipe](https://www.computerhope.com/jargon/p/pipe.htm) it to the **while** loop, where **read** reads the first word into **word1**, the second into **word2**, and everything else into **word3**. Output:

word1: one

word2: two

word3: three four

echo "one two three four" | while read -a wordarray; do

echo ${wordarray[1]}

done