Click to verify

echo [option] [string] The echo command is used to show a line of text or a variable's value in the terminal. Basic Usage To display a simple message, use echo "message": echo "Hello, World!" Hello, World! Options Overview The echo command has several options to customize its output: -n - Don't add a new line at the end - e - Allow special characters like for new lines -E - Don't allow special characters (default) Options: No Trailing Newline The -n option prevents echo from adding a newline at the end of the output. This is useful when you want to continue output on the same line. echo -n "Hello," ;echo " World!" Hello, World! The - e option enables the use of backslash escapes like for new lines, \t for tabs, etc. This allows for more formatted output. echo -e "HelloWorld!" Hello World! Options: Disable Backslash Escapes The - E option disables the use of backslash escapes, which is the default behavior. This ensures that the text is output exactly as typed. echo -E "HelloWorld!" HelloWorld! Using echo in Scripts The echo command is often used in scripts for debugging or logging information. It helps you see what's happening in your script by printing messages to the terminal. #!/bin/bash echo "Starting the script..." # Your script commands here echo "Script finished." ECHO(1) User Commands ECHO(1) echo - display a line of text echo [SHORT-OPTION]... [STRING]... echo LONG-OPTION Echo the STRING(s) to standard output. -n do not output the trailing newline - e enable interpretation of backslash escapes - E disable interpretation of backslash escapes (default) --help display this help and exit --version output version information and exit If -e is in effect, the following sequences are recognized: \ backslash \a alert (BEL) \b backspace \ c produce no further output \e escape \f form feed new line \r carriage return \t horizontal tab \v vertical tab 0NNN byte with octal value NNN (1 to 3 digits) \xHH byte with hexadecimal value HH (1 to 2 digits) Your shell may have its own version of echo, which usually supersedes the version described here. Please refer to your shell's documentation for details about the options it supports. Consider using the printf(1) command instead, as it avoids problems when outputting option-like strings. Written by Brian Fox and Chet Ramey. GNU coreutils online help: < Report any translation bugs to < Copyright 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. printf(1) Full documentation < or available locally via: info '(coreutils) echo invocation' This page is part of the coreutils (basic file, shell and text manipulation utilities) project. Information about the project can be found at ⟨ ��. If you have a bug report for this manual page, see ⟨ ��. This page was obtained from the tarball coreutils-9.6.tar.xz fetched from ⟨   on 2025-02-02. If you discover any rendering problems in this HTML version of the page, or you believe there is a better or more up-to-date source for the page, or you have corrections or improvements to the information in this COLOPHON (which is not part of the original manual page), send a mail to man-pages@man7.org Pages that refer to this page: ldapcompare(1), systemd-ask-password(1), systemd-run(1), cpuset(7) The echo command is a built-in feature in Linux that prints out its arguments as standard output. It is used to display text strings or the command results. This tutorial explains different ways to use the echo command in Linux through various examples. Prerequisites A system running Linux (this tutorial uses Ubuntu 22.04). Access to the terminal. The echo command in Linux displays a string provided by the user. The echo command syntax is: echo [option] [string] The echo command has several arguments. The following table presents commonly used echo command options. Option Description n Displays the output while omitting the newline after it. - E The default option. Disables the interpretation of escape characters. - e Enables the interpretation of escape characters. --help Displays a help message with information about the echo command and its options. --version Prints the echo command version information. The echo command prints text or variables in the terminal. It's commonly used in scripts and command-line operations to provide feedback, print messages, or output variable values. The following text presents ways to use the echo command in Linux. Run the following command to print Hello, World! as the output: echo Hello, World! Using echo without any options prints a string as is, without any changes to the output. The - e option is used with escape characters, as it enables their use in the output. The escape characters are useful for formatting output and adding special characters or effects to text displayed by the echo command. Escape Character Description \\ Displays a backslash character. \a Plays a sound alert when displaying the output. \b Removes all the spaces between the text. \ c Omits any output following the escape character. Adds a newline character to the output, which signifies the end of one line of text and the beginning of a new line. \r Performs a carriage return, which moves the cursor to the beginning of the current line without advancing to the next line. \t Creates horizontal tab spaces. \v Creates vertical tab spaces. For instance, using \ c lets you shorten the output by omitting the part of the string that follows the escape character: echo -e 'Hello, World! \c This is PNAP!' Note: If you are using the - e option, enter your string enclosed in single quotation marks. This ensures that escape characters are interpreted correctly. Use any time you want to move the output to a new line: echo -e 'Hello, World, this is PNAP!' Add horizontal tab spaces by using \ t : echo -e 'Hello, \tWorld!' Use \ v to create vertical tab spaces: echo - e 'Hello, \vWorld, \vthis \vis \vPNAP! ' Using ANSI escape sequences lets you change the output text color: echo -e '\033[1;37mWHITE' echo -e '\033[0;30mBLACK' echo -e '\033[0;31mRED' echo - e '\033[0;34mBLUE' echo -e '\033[0;32mGREEN' Use > or >> with the echo command to print the output to a file instead of displaying it in the terminal. If the specified text file doesn't already exist, this command creates it. Run the following: echo - eecho command used in Linux with examples The echo command is a vital tool for shell users that allows us to get visible output from shell scripts and can include variables, filenames, and directories. For instance, run: echo "Hello, world!" | tee output.txt To verify the output was also written to the file, run the following command: cat output.txt The echo command is used to display variable values as output. For instance, to display the current user name, use: echo $USER Another option is to declare multiple variables and then use the echo command to display its value. For example, run the following: echo "Hello, my name is $name and I am $age years old." The echo command allows you to include the result of other commands in the output. For instance, run the ls command to list all the files and directories in the Home directory by using: echo "This is the list of directories and files on this system: $(ls)" The -n option in the echo command omits the trailing newline added at the end of the output. For example: echo -n "Enter your name: " Here, echo -n "Enter your name: " prints the prompt without a newline, so the user's input appears on the same line as the prompt. The read command waits for the user to type something and press Enter. Whatever the user types is stored in the variable $name. The echo "Hello, $name!" command prints a greeting message that includes the $name variable value. Since we entered Sara at the prompt, the output is Hello, Sara!. Using echo before a potentially dangerous command, such as rm -rf, allows you to see what files and directories are affected without executing the command. For example, display the potential files and directories that match the pattern "f*" in the current directory without deleting them: echo rm -rf f* The echo command, without any options, prints the names of all files and directories in the mentioned directory. It behaves similarly to the ls command. See the current directory contents with: echo / Use echo to print only certain file types. For instance, print only .txt files in the current directory with: echo *.txt Okay, the user wants me to paraphrase the given article text using one of the three methods: spelling errors, non-native English speaker style, or increased burstiness. Let me first understand the original text. It's about using the echo command in Linux, explaining how to handle quotes, variables, and redirection. The user provided examples of each method, so I need to apply one randomly. First, I need to decide which method to use. Since the probabilities are 40% for SE, 30% for NNES, and 30% for IB, I'll randomly select one. Let me pick the spelling errors method (SE) for this example. Now, I'll go through the original text and introduce occasional spelling mistakes. For instance, "reading — and debugging — scripts much easier" could become "readin — and debuggin — script much easier." I need to ensure the errors are rare and don't disrupt readability. Next, the part about including a double quote: "just put a backslash \ in front of the double quote mark (with no space between them)." Maybe change "backslash" to "backslash" (no change here) or adjust spacing. Wait, the example shows "put a backslash \ in front" becomes "put a backslash \ in front" with no space. Maybe leave that as is. Looking at the example where they use echo with variables: "my_name="Dave"" could become "my_name="Dave"" with a typo in "Dave" like "Daves". But the original example had "Dave" as correct. Maybe change "Dave" to "Daves" to introduce a spelling error. Another part: "echo 'My name is $my_name'" could become "echo 'My name is $my_name'" with a typo in "my_name" like "my_name" (no change), or maybe "my_name" becomes "my_name" with a missing apostrophe. Wait, the original has "my_name" correctly. Maybe change "echo" to "echoo" as a typo. Wait, the user's example for SE had "Lookin forward to seein everyone..." So I need to make similar minor errors. Let me check each sentence and introduce 1-2 spelling errors. For example, "That's easy, just put a backslash \ in front of the double quote mark (with no space between them)." becomes "That's easy, just put a backslash \ in front of the double quote mark (with no space between them)." No changes here. Maybe "double quote" becomes "double quate" with a missing 't'. Another line: "echo -e "This is a long line of textsplit across three lineswith\ttabs\ton\tthe\tthird\tline"" could have "textsplit" as "textsplit" (no change) or "textsplit" becomes "textsplit" (no typo). Wait, maybe "textsplit" becomes "textsplit" (no typo). Hmm, perhaps "textsplit" is correct. Maybe "lineswith" becomes "lineswith" (no typo). Wait, maybe "textsplit" should be "text split" but the original has it as one word. So maybe leave it. Alternatively, "textsplit" could be "textsplit" (no change). This is a bit tricky. I need to ensure that the errors are minor and not too disruptive. Let me go through each part again and make sure I add 1-2 spelling errors without making the text unreadable. Once done, I'll review the paraphrased text to ensure it's still understandable with the added errors. Then, I'll present it as the final answer. Here’s the paraphrased text using the **spelling errors** method (40% probability): --- **Readin — and debuggin — script much easier**. To include a double quote, just put a backslash `\` in front of the double quote mark (with no space between them). This is important when working with variables, like `my_name="Daves"`, where the value is stored. When using `echo`, you can write: `echo 'My name is $my_name'` to display the variable. If you want to handle special characters, use the `-e` option. For example: `echo -e "This is a long line of textsplit across three lineswith\ttabs\ton\tthe\tthird\tline"`. The `\t` adds tabs, and `\n` creates new lines. To avoid newline issues, use `-n`: `echo -n "no final newline"`. This keeps the command prompt right after the text. Redirection is also useful. Use `> sample.txt` to create a file, or `>> sample.txt` to append to it. For example: ```bash echo "Creating a new file." > sample.txt echo "Adding to the file." >> sample.txt cat sample.txt ``` This creates a log file with timestamps: `echo "Logfile started: $(date +'%D %T')" > logfile.txt`. The `echo` command is a built-in tool in Linux, used for outputting text or variables. Its syntax is simple but powerful. --- *Note: Minor spelling errors (e.g., "readin" instead of "reading," "Daves" instead of "Dave") were introduced to reflect the SE method while maintaining readability.*echo command in Linux provides various options to display text or strings. 1. -e here enables backslash escapes Example: ``` echo -e "Geeks \bfor \bGeeks" ``` removes space 2. \c suppress trailing new line with backspace interpreter ‘-e’ Example: ``` echo -e "Geeks \cfor Geeks" ``` continue without emitting new line in the above example, text after `\c` is not printed and omitted trailing new line. 3. : creates a new line from where it is used Example: ``` echo -e "Geeks for Geeks" create new line ``` 4. \t creates horizontal tab spaces Example: ``` echo -e "Geeks \tfor \tGeeks" creating horizontal tab space ``` 5. \r : carriage return with backspace interpreter ‘-e’ to have specified carriage return in output Example: ``` echo -e "Geeks \rfor Geeks" ``` Carriage return in the above example, text before `\r` is not printed. 6. \v creates vertical tab spaces Example: ``` echo -e "Geeks \vfor \vGeeks create vertical tab spaces ``` 7. \a : alert return with backspace interpreter ‘-e’ to have sound alert Example: ``` echo -e "\aGeeks for Geeks" ``` 8. echo * : prints all files/folders, similar to ls command Example: ``` echo * ``` 9. -n: omits echoing trailing newline Example: ``` echo -n "Geeks for Geeks" ``` 10. Redirecting `echo` Output The output of the `echo` can be redirected to a file instead of displaying it on the terminal. We can achive this by using the `>` or `>>` operators for output redirection Example: ``` echo "Welcome GFG" > output.txt ``` if `[“echo -n” = "-n"]; then n="" c=\c else n="-n" c="" fi echo ${n} Enter your name: ${c} read name echo "Hello, $name"