Continue

A "bash list" refers to a collection of commands or items managed in a shell script or an interactive session, where each item can be accessed sequentially.Here's a simple example of creating and displaying a list of items in bash:# Define a list of fruitsfruits=("apple" "banana" "cherry")# Loop through the list and print each fruitfor fruit in "${fruits[@]}"; do echo $fruitdoneUnderstanding Lists in BashWhat are Lists in Bash?In Bash, a list refers to a collection of items held together as a single entity. Lists are crucial for organizing and managing data efficiently in scripts. They allow you to store multiple values and manipulate them conveniently, making your scripts adaptable and powerful.Types of Lists in BashThere are two primary types of lists in Bash: arrays and associative arrays. Arrays are indexed collections of elements, whereas associative arrays allow you to link keys to values.Arrays: A standard array uses numerical indices to access its elements.Associative Arrays: These arrays pair keys (strings) with corresponding values, enhancing the clarity of data representation.Bash List Directories: Navigate Your Files Like a ProCreating Lists in BashDefining an ArrayCreating a traditional array is straightforward. You use parentheses to define the elements.my_array=(value1 value2 value3)With this syntax, you have an array named `my_array` containing the elements `value1`, `value2`, and `value3`.Defining an Associative ArrayTo create an associative array, you first need to declare it with the `declare -A` syntax.declare -A my_assoc_array=(["key1"]="value1" ["key2"]="value2")Here, `my_assoc_array` is defined, linking `key1` to `value1` and `key2` to `value2`.Adding Items to ListsYou can append new elements to an array using the `+=` operator. This method allows for dynamic modification of your lists.my_array+=("value4")After executing this line, `my_array` will contain `value1`, `value2`, `value3`, and `value4`.Bash List Users: A Quick Guide to User ManagementAccessing and Manipulating Bash ListsAccessing Elements in an ArrayTo access specific elements within an array, you use the index of the element.echo ${my_array[0]} # Outputs value1This command retrieves the first element in `my_array`, demonstrating how intuitive accessing list elements can be.Looping Through ArraysYou can iterate through all items in an array with a `for` loop. This is useful when you need to perform actions on each element of your list.for item in "${my_array[@]}"; do echo $itemdoneBy using `"${my_array[@]}"`, you get all elements, allowing comprehensive manipulation.Accessing Elements in an Associative ArraySimilarly, you can access elements in an associative array using their keys.echo ${my_assoc_array["key1"]} # Outputs value1This command showcases the versatility of associative arrays, where you can reference values via human-readable keys.Looping Through Associative ArraysIterating through an associative array involves looping over its keys.for key in "${!my_assoc_array[@]}"; do echo "$key: ${my_assoc_array[$key]}"doneThis loop provides both keys and values, making it easy to display or process data stored in the array.Bash List Environment Variables: A Quick GuideCommon Operations on Bash ListsRemoving Items from ArraysWhen you need to remove specific elements from an array, the `unset` command is your go-to solution.unset my_array[1] # removes value2Remember that after using `unset`, the index remains in the array, which can result in gaps. Careful management of array indices is essential.Sorting ArraysIf you want to sort the elements of an array, you can use the `sort` command in a combination with a subshell.sorted_array=($(for i in "${my_array[@]}"; do echo "$i"; done | sort))This snippet creates a new array, `sorted_array`, containing sorted elements.Combining ArraysMerging arrays can be accomplished using straightforward concatenation.combined_array=("${my_array[@]}" "${another_array[@]}")After this command, `combined_array` will include all the elements from both original arrays, enhancing your data management flexibility.Bash List Files in Directory: A Quick GuideBest Practices for Using Lists in BashChoosing Between Array TypesWhen deciding which type of array to use, consider the nature of your data. If you need simple indexed access, a regular array suffices. However, for datasets requiring meaningful keys, associative arrays improve readability and accessibility.Avoiding Common PitfallsA frequent error occurs when using uninitialized indices, which can lead to unexpected behavior. Make sure to initialize arrays and handle the presence of empty values to prevent complications.Unlocking Bash History: Navigate Your Command Line MemoryConclusionRecap of Key PointsIn summary, mastering bash listsboth traditional and associativedelivers powerful tools for data management in your scripts. The capacity to create, access, and manipulate lists enables greater flexibility and efficiency.Further Learning ResourcesFor those eager to expand their knowledge, various online courses, documentation, and blogs focus specifically on Bash scripting and list manipulation. Engaging with these resources will deepen your understanding and expertise.Call to ActionDon't hesitate to explore creating your own bash lists today! Share your experiences or examples in the comments section; your insights can foster a rich community of learning and growth.How to create indexed and associated arrays add, remove get iterate loop first and last elements of array s syntax and array cheat sheet shell examples.Arrays in shells are variable to hold more than one value.Suppose, You have a list of numbers 1 2 3.. 10 and want to store these numbers in Shell ScriptWithout arrays, You have to declare as followslet number1=1let number1=1.....let number10=10Iteration is difficult and if we want to store 100 numbers, It is very difficult.So, You can use an array referring to a single variable and store it.How to declare and create an array?There are two types of arrays we can createindexed arrays: array elements are stored with the index starting from zeroassociated arrays: array is stored with key-value pairsDeclare an arrayTo create an array, We need to declare an array.declares -a array; # indexed arraydeclare -A array; # associative arrayan array is declared with the keyword declare with option -a or Aindexed array exampleIn this, Array values are stored with index=0 onwards. these are created with declare and -a optiondeclare -a arrayarray=(one two three)This array is a store with index=0, incremented by 1 as follows.array[0]=onearray[1]=twoarray[2]=threeassociative array exampleIn this, Array values are stored with keys. these are created with declare and -A optiondeclare -A arrayarray=(one two three)In this array is a store with index=0, incremented by 1 as followsarray[key1]=onearray[key2]=twoarray[key3]=threeLets assign the values.array=(1,2,3,4)Assign the values without declaring an arrayarrayvariable[index]=valueThis means, that arrayvariable is declared and assigned an array index with value.Arrays are zero-indexed based on zero to the length of an array -1index=0 - returns the first elementindex=-1 returns the last elementArrays can contain numbers, strings, and a mix of itLets create an array of examples.Access the array valuesAn array contains an index to get elements.Array elements can be accessed using the below syntax.${array_name[index]}Declare an Array of numbers and loop throughArrays can contain numbersThis example contains an array of numbers and for loop to printnums=(1 3 12)for i in "${nums[@]}"do echo "$i"doneOutput:1312Declare an Array of strings and loop throughArrays can contain numbersThis example contains an array of numbers and for loop to printnumbers=("element1" "element2" "element3")for i in "${numbers[@]}"do echo "$i"doneOutput:element1element2element3Access the first elements of an arrayIn Array elements, the First element index is zero, and array[0] returns the first elementnumbers=("element1" "element2" "element3")echo ${numbers[0]}echo ${numbers}Output:element1element1Get the last element of an arrayIn a bash script, you can use index=-1 to get the last array element.numbers=("element1" "element2" "element3")echo ${numbers[-1]}With the recent bash 4.0 version, you can use the below syntax to read the last element.echo ${numbers[${#numbers[@]}-1]}Iterate or loop array elementsFor loop is used to iterate elements.Here is an example loop array example to print all elementsnumbers=("element1" "element2" "element3")for i in "${numbers[@]}"do echo "$i"doneOutput:element1element2element3Another way to print the index and elements of an array using for loop.numbers=("element1" "element2" "element3")for i in "${!numbers[@]}"do echo "$i" "${numbers[$i]}"doneOutput:0 element11 element22 element3Print all array elementsUse [@] or [*] to print all elements of an array.arr=("element1" "element2" "element3") //echo ${arr[@]} #element1 element2 element3echo ${arr[*]} #element1 element2 element3You can remove an element from an array using unset for a given index.numbers=("element1" "element2" "element3")echo ${numbers[*]}unset numbers[-1]echo ${numbers[*]}Adding an element to an arrayYou can add an element at any index position using the below syntax.array[index]=valueAn example of adding elements starting and end as well as middlenumbers=("element1" "element2" "element3")echo ${numbers[*]} numbers[0]="element0"echo ${numbers[*]} numbers[5]="element5"echo ${numbers[*]} numbers[6]="element6"echo ${numbers[*]}Output:element1 element2 element3element0 element2 element3element0 element2 element3 element5element0 element2 element3 element5 element6Length of an arrayIn this, Find the count of all elements in an array.Shell script provides #arr=("element1" "element2" "element3")echo ${#arr[@]} # returns 3echo ${#arr[*]} # returns 3Array cheat sheet examplesExampleDescriptiondeclare -a arrayDeclare an Indexed arraydeclare -A arrayDeclare an Associative arraydeclare -a array=()Declare an indexed array with empty arrayarray=()create an empty array with declaring is validarray=(1 6 3)Initialize array with numbersarray=(one two three)Initialize the array with stringarray=(one two 1)Initialize the array with mixed data${array[0]}Get first element${array[1]}Get Second element${array[-1]}Get Last element${array[@]}Get All elements${array[*]}Get All elements${!array[!]}Get All indexes${#array[!]}Array lengtharray[0]=12Add element to array at first position.i.e index=0array[-1]=22Add element to array at last position.array+=(11)Append value to an array${array[@]:k:i} Get index=1 element starting from index=k Arrays to the rescue!So far, you have used a limited number of variables in your bash script, you have created a few variables to hold one or two filenames and usernames. But what if you need more than a few variables in your bash scripts; lets say you want to create a bash script that reads a hundred different inputs from a user, are you going to create 100 variables? Luckily, you dont need to because arrays offer a much better solution.Creating your first array in a bash scriptLets say you want to create a bash script timestamp.sh that updates the timestamp of five different files. First, use the nave approach of using five different variables:#!/bin/bashfile1="f1.txt"file2="f2.txt"file3="f3.txt"file4="f4.txt"file5="f5.txt"touch $file1touch $file2touch $file3touch $file4touch $file5Now, instead of using five variables to store the value of the five filenames, you create an array that holds all the filenames, here is the general syntax of an array in bash:array_name=(value1 value2 value3)So now you can create an array named files that stores all the five filenames you have used in the timestamp.sh script as follows:files=("f1.txt" "f2.txt" "f3.txt" "f4.txt" "f5.txt")As you can see, this is much cleaner and more efficient as you have replaced five variables with just one array!Accessing array elements in bashThe first element of an array starts at index 0 and so to access the nth element of the array you use the n -1 index. For example, to print the value of the 2nd element of your files array, you can use the following echo statement:echo ${files[1]}and to print the value of the 3rd element of your files array, you can use:echo ${files[2]}and so on.The following bash script reverse.sh would print out all the five values in your files array in reversed order, starting with the last array element:#!/bin/bashfiles=("f1.txt" "f2.txt" "f3.txt" "f4.txt" "f5.txt")echo ${files[4]}echo ${files[3]}echo ${files[2]}echo ${files[1]}echo ${files[0]}I know you might be wondering why so many echo statements and why don't I use a loop here. This is because I intend to introduce bash loop concepts later in this series.You can also print out all the array elements at once:echo ${files[*]}f1.txt f2.txt f3.txt f4.txt f5.txtYou can print the total number of the files array elements, i.e. the size of the array:echo ${#files[@]}5You can also update the value of any element of an array; for example, you can change the value of the first element of the files array to a.txt using the following assignment:files[0]="a.txt"Adding array elements in bashLets create an array that contains the name of the popular Linux distributions:distros=("Ubuntu" "Red Hat" "Fedora")The distros array current contains three elements. You can use the += operator to add (append) an element to the end of the array. For example, you can append Kali to the distros array as follows:distros+=("Kali")Now the distros array contains exactly four array elements, with Kali being the last element of the array.Deleting array elements in bashLets first create a num array that will store the numbers from 1 to 5:num=(1 2 3 4 5)You can print all the values in the num array:echo ${num[*]}1 2 3 4 5You can delete the 3rdelement of the num array by using the unset shell built-in:unset num[2]Now, if you print all the values of the num array:echo ${num[*]}1 2 4 5As you can see, the third element of the array num has been deleted.You can also delete the whole num array in the same way:unset numCreating hybrid arrays with different data typesIn bash, unlike many other programming languages, you can create an array that contains different data types. Take a look at the following user.sh bash script:#!/bin/bashuser=("john" 122 "sudo,developers" "bash")echo "User Name: ${user[0]}"echo "User ID: ${user[1]}"echo "User Groups: ${user[2]}"echo "User Shell: ${user[3]}"Notice the user array contains four elements:"John" ----> String Data Type122 ---> Integer Data Type"sudo,developers" ---> String Data Type"bash" ---> String Data TypeSo, its totally ok to store different data types in the same array. Isn't that awesome?This takes us to the end of this tutorial; I hope you enjoyed it! If you want something more complicated and real-world example, checkout how to split strings in bash using arrays.And, of course, you may practice what you just learned by solving the problems and referring to their solutions if you get stuck or need a hint. In the next chapter, I will show you how to use various bash arithmetic operators. In Bash, there is no data type named list, but you can generate a list using bash scripts that you can use for your needs. Also, sometimes associative arrays are referred to as lists in Bash scripting. In this article, I will demonstrate how to create a list in Bash scripts and give some examples. Practice Files to Create List in Bash2 Methods to Create List Using Bash ScriptsIn this article, you are going to learn how to use loops and arrays to generate lists using Bash script. You can use either of the methods to create a list. Also, you will notice that while using an array, I have written all the contents of the list inside the array. On the other hand, when I used loops, the list was generated by the loop. Method 1: Create a List Using Loops in Bash ScriptsIn this method, you will see how to generate a list using for and while loops. Also, I have tried to make this list generation interactive, so that users can use it according to their needs.Case 1: Generate List Using "for" Loop in Bash ScriptSuppose, I want to list files with employee in the name. To do so Ive used for loop to go through the files in loops. If you want to achieve this, please check & execute the following Bash script:#!/bin/bash#listing all files named employeefor a in $(ls employee*)do ls -l "$a"doneThe first line, #!/bin/bash is called shebang, which specifies the interpreter as Bash, which will be used to execute the script. Then, the for loop iterates over the list of files in the current directory that contain the name employee and the output is captured and processed by the loop. After that, the line ls -l "$a" basically executes the ls -l command for each file captured by the loop and displays detailed information about the file, including its permission, owner, size, and modification timestamp. Finally, the $a is to represent the current file being processed in each iteration.After running the script, it printed the list of files containing employees in them.Case 2: Generate List Using "while" Loop in Bash ScriptIn this case, I will show you how to generate a list of numbers using a while loop. Also, to make this script more interactive, the list of numbers will take user input. So, you can get the list of numbers according to your choice.To list numbers with a while loop, see the below bash script:#!/bin/bash#Prompt user to enter the number of iterationsread -p "Enter the number of iterations: " total_iterationsi=1 #current iteration# Start the loopwhile [$i -le $total_iterations]do echo "$i" # Print the current iteration ((i++)) # Increment the counter variabledoneThe script prompts the user to enter the number of iterations and store it in the variable named total_iterations. Then i=1 is used to declare and initialize the variable i to 1 which represents the current iteration. After that, a while loop continues until the value i is less than or equal to the total_iterations. The ((i++)) increments the value of i by 1, moving to the next iteration. Finally, the echo command prints the value of every iteration until the end of the loop.After running the script, a user prompt takes the iteration number from the user. According to the inserted number, the script prints numbers. For example, as I entered 7, it showed a list of numbers from 1 to 7.Method 2: Create a List Using Arrays in Bash ScriptsIn this method, I have demonstrated the process of generating any list using an array. I have generated a list of months. Moreover, you can use the array to generate any list you want, you just have to change the list's contents.To list the names of months using arrays, go through the following bash script:#!/bin/bash# Declare an array of month namesMonths=("January" "February" "March" "April" "May" "June" "July" "August" "September" "October" "November" "December")# Print the list of monthsecho "The list of Months: ${Months[@]}"This script declares an array named Months, which contains the names of months. Finally, the echo command prints the list of months and ${Months[@]} expands the array to include all elements.After running the script, it prints the list of months given inside the array.Comparative Analysis of Methods to Generate List in BashIn this section, I will give you a comparative analysis of the two methods mentioned above so you can understand which will be best for you to use.MethodsProsConsMethod 1The list is automatically generated by loops.Those who have little to no knowledge of programming can find it complicated to write scripts using loops.Method 2If you generate a list using arrays, it is relatively simple, as you just have to know array syntax.The contents of the list have to be given inside the array.If you want to generate a list automatically, you better use the first method, but you must have a firm understanding of conditional loops. On the other hand, if you already have a list and want to manipulate it and print it in different ways, you should go for the second method.ConclusionSo far, you've learned to generate a list with Bash scripting. Using loops or arrays, you can now quickly create lists. Choose the one that best suits your needs. Feel free to ask any questions regarding this article.People Also AskWhat is the command to list in bash?The ls command is used to list in Bash. This command is specifically used for making a list of directories and files.How do you define a list in a Bash script?A list is an associative array in Bash scripting. So, defining a list means declaring an array in Bash. The syntax to declare an array in Bash is array_name={element1 element2....elementN}.How do I print all list elements in Bash?To print all elements of a Bash array, you can use the declare command with the command option -p. So, the syntax is declare -p Array_name.How to generate UUID in shell script?To generate UUID in a shell script, you can use the uuidgen command. This command is typically available in Unix-like systems, including Linux. For that, use the syntax, my_uuid=$(uuidgen).How do I list all users in Bash?To list all users in Bash, you can use the cat command along with /etc/passwd. So, this passwd file in the etc directory contains all user names and when you execute this, you can see all user names printed on your screen. Related Articles