<- Commands and Arguments | Parameters ->

Special characters

Some characters are evaluated by Bash to have a *non-literal* meaning. Instead, these characters carry out a special instruction, or have an alternate meaning; they are called "special characters", or "meta-characters".

Here are some of the more common special characters uses:

Char.	Description
`" "`	*Whitespace* — this is a tab, newline, vertical tab, form feed, carriage return, or space. Bash uses whitespace to determine where words begin and end. The first word is the command name and additional words become arguments to that command.
`$`	*Expansion* — introduces various types of expansion: parameter expansion (e.g. `$var` or `${var}`), command substitution (e.g. `$(command)`), or arithmetic expansion (e.g. `$((expression))`). More on expansions later.
`' '`	*Single quotes* — protect the text inside them so that it has a *literal* meaning. With them, generally any kind of interpretation by Bash is ignored: special characters are passed over and multiple words are prevented from being split.
`" "`	*Double quotes* — protect the text inside them from being split into multiple words or arguments, yet allow substitutions to occur; the meaning of most other special characters is usually prevented.
`\`	*Escape* — (backslash) prevents the next character from being interpreted as a special character. This works outside of quoting, inside double quotes, and generally ignored in single quotes.
`#`	*Comment* — the `#` character begins a commentary that extends to the end of the line. Comments are notes of explanation and are not processed by the shell.
`=`	*Assignment* -- assign a value to a variable (e.g. `logdir=/var/log/myprog`). Whitespace is *not* allowed on either side of the `=` character.
`[[ ]]`	*Test* — an evaluation of a conditional expression to determine whether it is "true" or "false". Tests are used in Bash to compare strings, check the existence of a file, etc. More of this will be covered later.
`!`	*Negate* — used to negate or reverse a test or exit status. For example: `! grep text file; exit $?`.
`>, >>, <`	*Redirection* — redirect a command's *output* or *input* to a file. Redirections will be covered later.
`|`	*Pipe* — send the output from one command to the input of another command. This is a method of chaining commands together. Example:

	`echo "Hello beautiful."	grep -o beautiful`.
`;`	*Command separator* — used to separate multiple commands that are on the same line.
`{ }`	*Inline group* — commands inside the curly braces are treated as if they were one command. It is convenient to use these when Bash syntax requires only one command and a function doesn't feel warranted.
`( )`	*Subshell group* — similar to the above but where commands within are executed in a subshell (a new process). Used much like a sandbox, if a command causes side effects (like changing variables), it will have no effect on the current shell.
`(( ))`	*Arithmetic expression* — with an arithmetic expression, characters such as `+`, `-`, `*`, and `/` are mathematical operators used for calculations. They can be used for variable assignments like `(( a = 1 + 4 ))` as well as tests like `if (( a < b ))`. More on this later.
`$(( ))`	*Arithmetic expansion* — Comparable to the above, but the expression is replaced with the result of its arithmetic evaluation. Example: `echo "The average is $(( (a+b)/2 ))"`.
`*, ?`	*Globs* -- "wildcard" characters which match parts of filenames (e.g. `ls *.txt`).
`~`	*Home directory* — the tilde is a representation of a home directory. When alone or followed by a `/`, it means the current user's home directory; otherwise, a username must be specified (e.g. `ls ~/Documents; cp ~john/.bashrc .`).
`&`	*Background* -- when used at the end of a command, run the command in the background (do not wait for it to complete).

Examples:

```
$ echo "I am $LOGNAME"
I am lhunath
$ echo 'I am $LOGNAME'
I am $LOGNAME
$ # boo
$ echo An open\ \ \ space
An open    space
$ echo "My computer is $(hostname)"
My computer is Lyndir
$ echo boo > file
$ echo $(( 5 + 5 ))
10
$ (( 5 > 0 )) && echo "Five is greater than zero."
Five is greater than zero.
```

Deprecated special characters (recognized, but not recommended)

This group of characters will also be evaluated by Bash to have a *non-literal* meaning, but are generally included for backwards compatibility only. These are not recommended for use, but often appear in older or poorly written scripts.

Char.	Description
` ` ` `	*Command substitution* - use `$(  )` instead.
`[  ]`	*Test* - an alias for the old `test` command. Commonly used in POSIX shell scripts. Lacks many features of `[[  ]]`.
`$[  ]`	*Arithmetic expression* - use `$((  ))` instead.

Additionally:

- **In The Manual —   Shell Syntax**
- *Special Characters* — Characters that have a special meaning to Bash. Usually their meaning is interpreted and then they are removed from the command before executing it.

<- Commands and Arguments | Parameters ->

BashGuide/SpecialCharacters (last edited 2019-02-14 20:04:40 by GreyCat)