# Right CASE

|  |
| --- |
| cp $file $destination # WRONG  cp -- "$file" "$destination" # Right  When in doubt, **double-quote every expansion** in your shell commands. |

# TYPES OF QUOTING

|  |
| --- |
| There are **three standard types of quotes** (or four if you count backslash escaping), **and two nonstandard Bash extensions**. This is the brief description of each of them. Read below for the [verbose version](http://mywiki.wooledge.org/Quotes#Verbose).   * **Single quotes**: '...' removes the special meaning of every character between the quotes. Everything inside single quotes becomes a literal string. The only character that you can't safely enclose in single quotes is a single quote. * **Double quotes**: "..." *prevents some substitutions but allows others*.   - Every substitution that begins with a dollar sign $ is performed,  - as is the legacy `...` (backtick) [command substitution](http://mywiki.wooledge.org/CommandSubstitution).  - Backslash \ escaping is also performed.  - *No word splitting or filename expansion* is performed.   * **Backticks**: `...` is the legacy command substitution syntax; deprecated in favor of $(...) but still permitted for historical reasons. See [FAQ 082](http://mywiki.wooledge.org/BashFAQ/082) for details. * **Backslash**: Putting \ in front of a metacharacter removes its special meaning. This works inside double quotes, or in the absence of quotes. It does not work inside single quotes. * **$'...'**: This is a Bash extension. It prevents everything except backslash escaping, and also allows special backslash escape sequences like \n for newline, \t for tab, and \xnn for bytes specified in hexadecimal. * **$"..."** : This is another Bash extension. It is used for [localization support](http://mywiki.wooledge.org/BashFAQ/098) and will not be covered on this page. |

|  |
| --- |
| Example:  $ foo=bar  $ echo '!%$\*&'"$foo"  !%$\*&bar |

# Effects of Quoting

## Preserve unescaped metacharacters

|  |
| --- |
| A shell command is parsed by the shell into words, using whitespace (regardless of the [IFS](http://mywiki.wooledge.org/IFS) variable) and other shell metacharacters.  **The first function of quoting is to permit words to contain these metacharacters.**  echo '&'   * Without quotes, the & would put the echo command [into the background](http://mywiki.wooledge.org/ProcessManagement). * With quotes, the & is simply made into a word, and passed as an argument to the echo command instead. |

## Prevent field splitting and ignore glob pattern characters

|  |
| --- |
| **The second purpose of quoting is to prevent “**[**word splitting**](http://mywiki.wooledge.org/WordSplitting)**” and “**[**globbing**](http://mywiki.wooledge.org/glob)**”.** .  cp -- "$filename" "$destination"  In this example, the double quotes protect the value of each parameter (variable) from undergoing word splitting or globbing should it happen to contain whitespace or wildcard characters (\* or ? or [...]).   * Without the quotes, a filename like hot stuff.mp3 would be split into two words, and each word would be passed to the cp command as a separate argument. * Or, a filename that contains \* with whitespace around it would produce one word for every file in the current directory. That is not what we want. * With the quotes, every character in the value of the filename parameter is treated literally, and the whole value becomes the second argument to the cp command |

## Expand argument lists

|  |
| --- |
| Double-quoting $@ or ${array[@]} has a special meaning.  "$@" expands to a list of words, with each positional parameter's value being one word.  Likewise, "${array[@]}"expands to a list of words, one per array element.  When dealing with the positional parameters or with the contents of an array as a list of words, always use the double-quoted syntax.  Double-quoting $\* or ${array[\*]} results in one word which is the concatenation of all the positional parameters (or array elements) with the first character of [IFS](http://mywiki.wooledge.org/IFS) between them. This is similar to the join function in some other languages, although the fact that you can only have a single join character can sometimes be a crippling limitation. |