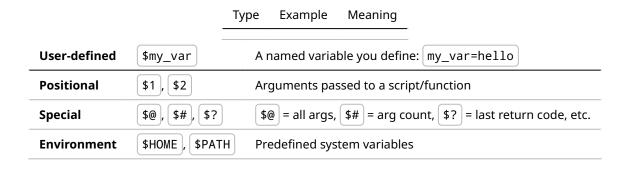
Bash Shell Variables, Quoting, Globbing, and Special Parameters

Nariable Types



Expansion Basics

Bash **expands variables** by replacing them with their values:

```
greeting="hello"
echo $greeting # → hello
```

CQuoting Rules

Context	Example	Behavior
Unquoted	echo \$var	Variables are expanded; word splitting and globbing happen.
Double-quoted	echo "\$var"	Variables are expanded; no word splitting or globbing. V Safe
Single-quoted	echo '\$var'	No expansion; literal string. 🗶 \$var is not replaced.

Examples

```
name="Tony"
echo Hello $name  # → Hello Tony
echo "Hello $name"  # → Hello Tony (safe from spaces or globbing)
echo 'Hello $name'  # → Hello $name (no expansion)

name="Tony Held"
```

```
echo Hello $name  # → Hello Tony Held ← splits words (can break args)
echo "Hello $name"  # → Hello Tony Held ← preserves as one word
```

Tip: Always quote unless you need word splitting.

```
mv "$source_file" "$destination_file" # Good
mv $source_file $destination_file # Bad if variables have spaces
```

Globbing (Filename Expansion)

Globbing matches filenames using wildcards:

Pattern	Matches
*	Zero or more characters ($file*$) \rightarrow $file1$, $fileA.txt$, etc.)
?	Exactly one character ($file?.txt$) \rightarrow $file1.txt$, $fileA.txt$)
[abc]	One character in set ($file[13].txt$) \rightarrow $file1.txt$, $file3.txt$)
[a-z]	Character range ($file[a-z].txt$) \rightarrow $filea.txt$, etc.)
**	Recursive glob (with shopt -s globstar)

```
echo *.txt  # Lists all .txt files
echo * # Lists everything
```

Globbing works in **unquoted** or **double-quoted** contexts; not in single quotes.

Special Bash Parameters

Parameter	Meaning	Use Case
\$0	Script name	echo "Script is \$0"
\$1 \$9	First to ninth argument	echo "First arg is \$1"
\${10}	Tenth argument	echo "Tenth arg is \${10}"
\$#	Number of arguments	echo "You passed \$# args"
\$@	All args as separate words	Best with quotes: "\$@"
\$*	All args as one string	Quoted: "\$*" = "arg1 arg2"

Parameter	Meaning	Use Case		
"\$@"	Expands to "arg1" "arg2"	Preserves args properly 🗸		
"\$*"	Expands to arg1 arg2"	Combines into one string X		
\$\$	PID of current script			
\$!	PID of last background command			
\$?	Exit code of last command			
\$_	Last argument to last command			
\$-	Current shell flags			

∥ \$@ vs \$* Examples

```
# Run: ./myscript.sh one "two words" three

# Inside the script:
echo $*  # one two words three
echo "$*"  # "one two words three" ← one long string
echo $@  # one two words three
echo "$@"  # "one" "two words" "three" ← correct separation
```

Summary Cheat Sheet

```
# Define & use a variable
var="hello"
echo $var
                 # → hello
echo "$var"
                 # → hello (safe)
echo '$var'
                 # → $var (no expansion)
# Special variables
$0
        → Script name
$1 .. $9 → Positional arguments
${10}
        → 10th arg
$#
         → Number of arguments
         → All args, separated (use "$@")
$@
$*
        → All args, combined (avoid)
$?
        → Exit status
        → PID of this script
$$
$!
        → PID of last background job
        → Last argument of last command
$_
```

```
# Globbing
         → Any number of characters
?
         → One character
         → a, b, or c
[abc]
         → Range
[a-z]
         → Recursive (needs `shopt -s globstar`)
# Quoting behavior
        → Expanded, splits & globs
$var
        → Expanded safely 🔽
"$var"
         → No expansion 🗙
'$var'
```

This file provides a concise and practical reference for understanding shell variables, quoting rules, globbing, and special parameter usage in Bash. Always quote variables unless you know you need expansion and word splitting.