BASH Variables

In Bash, variables are used to store values that you can reuse and manipulate throughout your script. They can hold numbers, strings, command outputs, and more.

1. Basic Syntax for Variables

To define a variable in Bash, you use this simple syntax:

```
variable_name=value
```

- **No spaces** should appear around the = sign.
- Variable names are case-sensitive and typically use uppercase by convention, but lowercase is also common.

Example:

```
NAME="Alice"
AGE=30
```

To access the value of a variable, prefix it with $:

```
echo "$NAME is $AGE years old."
```

2. Rules for Variable Naming

- Variable names must start with a letter or underscore.
- They can contain letters, numbers, and underscores.
- Avoid using special characters, as they may have special meanings in Bash.

Valid Examples:

```
MY_VAR="Hello"
_var2="Test"
```

Invalid Examples:

```
1name="John"   # Invalid: starts with a number
my-var="Doe"   # Invalid: uses a hyphen
```

3. Types of Variables

3.1 Local Variables

These are defined and used within the script or function they're declared in.

```
CITY="New York"
echo "The city is $CITY"
```

3.2 Environment Variables

Environment variables are accessible to child processes and the entire system session. They're usually declared with export.

```
export PATH_TO_FILES="/usr/local/files"
echo "$PATH_TO_FILES"
```

3.3 Special Variables

Bash has several built-in special variables:

- $0: The name of the script
- $1, $2, etc.: Positional parameters for command-line arguments
- $#: Number of arguments passed to the script
- $@: All arguments as separate words
- $?: Exit status of the last command
- $$: Process ID of the script

Example:

```
echo "Script name: $0"
echo "First argument: $1"
```

4. Assigning Values to Variables

4.1 Simple Assignment

As shown earlier, you can directly assign values to variables:

```
GREETING="Hello, World!"
```

4.2 Assigning Command Output

You can use **command substitution** to assign the output of a command to a variable using $(...):

```
CURRENT_DATE=$(date)
echo "Today's date is $CURRENT_DATE"
```

4.3 User Input

You can also assign values to variables based on user input using read:

```
echo "Enter your name:"
read USER_NAME
echo "Hello, $USER_NAME!"
```

5. Quoting Variables

- **Double quotes (")**: Preserves whitespace in variables but allows variable expansion.
- **Single quotes (')**: Prevents variable expansion, treating everything as a literal string.

Example:

```
NAME="Alice"
echo "Hello, $NAME"    # Expands $NAME
echo 'Hello, $NAME'    # Does not expand $NAME, outputs "Hello, $NAME"
```

6. Operations on Variables

6.1 String Operations

Concatenation: Combine strings directly.

```
FIRST="Hello"

SECOND="World"

GREETING="$FIRST $SECOND"

echo "$GREETING"
```

Length of a String:

```
STRING="Hello"

echo ${#STRING}  # Outputs: 5
```

6.2 Arithmetic Operations

Bash supports arithmetic operations using ((...)) or expr.

```
NUMBER=5

echo $((NUMBER + 3))  # Outputs: 8
```

```
echo $(expr $NUMBER + 3)  # Outputs: 8 (alternative method)
```

6.3 Incrementing and Decrementing

```
COUNTER=1

((COUNTER++))  # Increment by 1

echo "$COUNTER"  # Outputs: 2


((COUNTER--))  # Decrement by 1

echo "$COUNTER"  # Outputs: 1
```

8. Unsetting Variables

To delete a variable, use unset:

```
unset NAME

echo "$NAME"  # Outputs an empty line if NAME is unset
```

9. Example Script Using Variables

Here's a simple script that demonstrates the use of variables, user input, and command substitution.

```
#!/bin/bash

# Basic variable usage example

echo "Enter your name:"

read NAME

echo "Hello, $NAME!"


CURRENT_DATE=$(date)
```

```
echo "Today is $CURRENT_DATE"

echo "Enter your age:"

read AGE

NEXT_AGE=$((AGE + 1))

echo "Next year, you'll be $NEXT_AGE years old!"
```

This script:

1. Prompts the user for their name and age.
2. Greets the user.
3. Displays the current date.
4. Calculates the user's age next year and displays it.

Common Environment Variables:

Variable	Description	Example Value
`HOME`	User's home directory	`/home/user`
`PATH`	Directories to search for executable files	`/usr/local/bin:/usr/bin:/bin`
`USER`	Current logged-in user	`username`
`SHELL`	Default shell for the user	`/bin/bash`
`LANG`	Default language and localization setting	`en_US.UTF-8`
`PWD`	Current working directory	`/home/user/documents`
`OLDPWD`	Previous working directory	`/home/user/downloads`

Predefined variables

Command	Description	Example Usage
`$(date)`	Returns the current date and time.	`current_date=$(date)`
`$(pwd)`	Prints the current working directory.	`current_directory=$(pwd)`
`$(whoami)`	Returns the current logged-in user.	`user=$(whoami)`
`$(hostname)`	Returns the system's hostname.	`system_hostname=$(hostname)`
`$(ls)`	Lists files in the current directory (or directory passed as argument).	`files_list=$(ls)`
`$(df)`	Shows the disk space usage of the file systems.	`disk_usage=$(df)`
`$(free)`	Displays the system's memory usage.	`memory_usage=$(free -h)`
`$(top -n 1)`	Shows the top command output (system processes).	`top_output=$(top -n 1)`
`$(ps aux)`	Returns a list of running processes in a system.	`process_list=$(ps aux)`
`$(grep 'pattern' file)`	Searches for the string `pattern` in `file` and returns matching lines.	`search_result=$(grep 'pattern' myfile.txt)`
`$(wc -l filename)`	Returns the number of lines in a file.	`line_count=$(wc -l myfile.txt)`
`$(curl -s http://example.com)`	Fetches content from a URL and returns it.	`content=$(curl -s http://example.com)`
`$(echo $VARIABLE)`	Echoes the value of a variable.	`echo_output=$(echo $USER)`
`$(id -u)`	Returns the user ID of the current user.	`user_id=$(id -u)`
`$(stat file)`	Displays detailed information about a file, such as size, permissions, and modification time.	`file_stats=$(stat myfile.txt)`
`$(dig example.com)`	Queries DNS for information about a domain.	`dns_info=$(dig example.com)`

Summary:

Concept	Description	Example
Defining Variables	Assign values to variables with no spaces around `=`	`NAME="Alice"`
Accessing Variables	Use `$` prefix to access a variable's value	`echo "$NAME"`
Local Variables	Used within the script or function	`CITY="New York"`
Environment Variables	Accessible to child processes, declared with `export`	`export PATH_TO_FILES="/usr/local/files"`
Special Variables	Built-in variables like `$0` (script name), `$1` (first argument)	`$#` (argument count), `$$` (process ID)
Direct Assignment	Assign values directly	`GREETING="Hello"`
Command Output	Use `$(...)` for command substitution	`DATE=$(date)`
User Input	Capture user input with `read`	`read USER_NAME`
Double Quotes (")	Expands variables, preserves spaces	`echo "Hello, $NAME"`
Single Quotes (')	Treats everything literally, no variable expansion	`echo 'Hello, $NAME'`
String Concatenation	Combine strings directly	`FULL_NAME="$FIRST $LAST"`
String Length	Use `${#variable}` to get the string length	`echo ${#STRING}`
Arithmetic Operations	Use `$((...))` for calculations	`COUNT=$((COUNT + 1))`
Exporting Variables	Make a variable available globally with `export`	`export VAR_NAME="value"`
Unsetting Variables	Delete a variable with `unset`	`unset VAR_NAME`