SKILLS

Command Substitution and Process Substitution in Bash

Lesson Plan

Introduction

In Bash scripting, command substitution and process substitution are powerful features that allow you to manipulate and use the output of commands within your scripts. These features can make your scripts more efficient and flexible by allowing you to work with command outputs directly.

Table of Contents

Command Substitution

Command substitution allows you to use the output of a command as an argument in another command. This is achieved by enclosing the command in $(...) or using backticks `...`.

Syntax

Using $(...):
result=$(command)

Using backticks:
result=`command`

Examples

Storing Command Output in a Variable

```
# Using $(...)
current_date=$(date)
echo "Current date and time: $current_date"

# Using backticks
current_date=`date`
echo "Current date and time: $current_date"
```

Using Command Output in Another Command

```
# Listing files in the directory with the most recently
modified file
most_recent_file=$(ls -t | head -n 1)
echo "The most recent file is: $most_recent_file"
```

Process Substitution

Process substitution provides a way to pass the output of a command as if it were a file. It is useful when a command expects file input but you want to provide the output of another command.

Syntax

Using <(...):
command <(other_command)

Using >(...):
command >(other_command)

Examples
Comparing Files Using diff

```
# Comparing the output of two commands
diff <(ls /etc) <(ls /usr)
```

Using Process Substitution with sort

```
# Sorting the output of multiple commands
sort <(echo "apple") <(echo "banana") <(echo "cherry")
```

Comparison and Use Cases

- **Command Substitution:**
  - Ideal for capturing the output of a command to be used later in a script.
  - Commonly used in assignments, as arguments to commands, and in expressions.

- **Process Substitution:**
  - Best used when a command requires file input, and you want to provide the output of another command directly.
  - Useful in scenarios where commands need to compare or merge output streams.

Conclusion

Both command substitution and process substitution are valuable techniques in Bash scripting. Command substitution allows for the dynamic use of command output as variables or arguments, while process substitution enables commands to interact with the output of other commands as if it were file input. Mastering these features can greatly enhance your scripting capabilities and streamline your workflows.