Command Chaining and Redirection

Chaining and redirection are powerful features in Linux shell commands that allow users to combine commands and manage input/output effectively.

Chaining

Chaining allows you to run multiple commands in sequence. There are several ways to chain commands, each serving a different purpose:

1. Semicolon (;):

Executes multiple commands sequentially, irrespective of whether the previous command fails or succeeds. Failures in one command do not block the execution of subsequent commands.

`Syntax: command1; command2;……..;commandN;`

Example 1:

```
 amit@archlinux  ~/Test  echo "Chaining using Semicolon" ; ls -l;
Chaining using Semicolon
total 4
-rw-r--r-- 1 amit users    0 Dec 28 12:15 file1.txt
-rw-r--r-- 1 amit users    0 Dec 28 12:16 file2.txt
drwxr-xr-x 2 amit users 4096 Dec 28 12:16 mydir
```

Example 2:

```
 amit@archlinux  ~/Test  ls
file1.txt  file2.txt  mydir
 amit@archlinux  ~/Test  cd newdir; echo "This will be executed even if cd fails"
cd: no such file or directory: newdir
This will be executed even if cd fails
```

---

2. Logical And (&&):

Executes the next command only if the previous command succeeds i.e (returns an exit status of `0`). Failures in one command blocks the execution of subsequent commands.

`Syntax: command1 && command2 && …….&& commandN`

Example 3:

```
 amit@archlinux  ~/Test  echo "Chaining using Logical And" && mkdir  -v newdir
Chaining using Logical And
mkdir: created directory 'newdir'
```

Example 4:

```
 amit@archlinux  ~/Test  ls
file1.txt  file2.txt  mydir  newdir
 amit@archlinux  ~/Test  cd nodir && echo "This won't be executed"
cd: no such file or directory: nodir
 ✘ amit@archlinux  ~/Test
```

---

3. Logical Or (||):
Executes the next command only if the previous command fails i.e(returns a non-zero exit status).If the first command succeeds, the subsequent commands in the chain are skipped.

`Syntax: command1 || command2 || ……..|| commandN;`

Example 5:

```
 amit@archlinux  ~/Test  echo "Only This will be executed" || echo "This won't" || echo "This won't";
Only This will be executed
 amit@archlinux  ~/Test
```

Example 6:

```
 amit@archlinux  ~/Test  ls
file1.txt  file2.txt  mydir  newdir
 amit@archlinux  ~/Test  cd nodir ||  echo "This will be executed" || echo "This won't";
cd: no such file or directory: nodir
This will be executed
 amit@archlinux  ~/Test
```

---

4. Pipe (|):
Passes the output of one command as the input to another command. Commands are connected in a pipeline, where the standard output (`stdout`) of one command becomes the standard input (`stdin`) of the next. Failure of any command affects how subsequent command behaves.

`Syntax: command1 | command2 | …….| commandN;`

Example 7:

```
 amit@archlinux  ~/Test  cat month.txt | sort -M

Jan
Feb
Mar
Apr
```

Example 8:

```
 amit@archlinux  ~/Test  cat month.txt | sort -M | grep "Jan"
Jan
```

---

5. Logical Not (!):
Logical Not doesn't have a direct chaining operator like '&&' or '||'. Instead we can use '!' to reverse the exit status of a command.

`Syxtax: ! Commnd`

Example 9:

```
 amit@archlinux  ~/Test  ! cd nodir && echo "This will be executed";
cd: no such file or directory: nodir
This will be executed
```

Redirection
Redirection refers to the process of controlling where input, output and error messages of the command are sent. By default, commands take input from the keyboard (stdin) and send output or errors to the terminal (stdout and stderr). Redirection allows us to change these defaults.

File Descriptors

- **0**: Standard Input (`stdin`) – Input stream (default: keyboard).
- **1**: Standard Output (`stdout`) – Output stream (default: terminal).
- **2**: Standard Error (`stderr`) – Error messages stream (default: terminal).

1. Output Redirection (>):

Redirects standard output to a file, overwriting the file if it exists.

`Syntax: command > file` or `command 1> file`

Example 10:

```
 amit@archlinux  ~/Test  echo "Hello, World" > hello.txt
 amit@archlinux  ~/Test  ls
hello.txt
 amit@archlinux  ~/Test  cat hello.txt
Hello, World
 amit@archlinux  ~/Test
```

Example 11:

```
 amit@archlinux  ~/Test  echo "This will overwrite hello.txt" > hello.txt
 amit@archlinux  ~/Test  cat hello.txt
This will overwrite hello.txt
 amit@archlinux  ~/Test
```

---

2. Append Output (>>):
Redirects standard output to a file, appending the output to the file if it exists instead of overwriting.

`Syntax: command >> file` or `command 1>> file`

Example 12:

```
 amit@archlinux  ~/Test  echo "This is example for >>" >> file.txt
 amit@archlinux  ~/Test  cat file.txt
This is example for >>
 amit@archlinux  ~/Test
```

Example 13:

```
 amit@archlinux  ~/Test  echo "This line will be appended in file.txt" >> file.txt
 amit@archlinux  ~/Test  cat file.txt
This is example for >>
This line will be appended in file.txt
 amit@archlinux  ~/Test
```

---

3. Error Redirection (2> or 2>>):
Redirects error message to the file.

Syntax: command 2> file (this overwirtes if file exists) or
command 2>> file (this appends error if file exists)

Example 14:

```
 amit@archlinux  ~/Test  cd nodir 2> erro.txt
 ✘ amit@archlinux  ~/Test  cat erro.txt
cd: no such file or directory: nodir
 amit@archlinux  ~/Test
```

Example 15:

```
 amit@archlinux  ~/Test  wrong_command 2>> erro.txt
 ✘ amit@archlinux  ~/Test  cat erro.txt
cd: no such file or directory: nodir
zsh: command not found: wrong_command
```

---

4. Both Output and Error Redirection (&>):
Combines `stdout` and `stderr` into a single stream and redirects them together.

Syntax: command > file 2>&1 or
command &> file (in modern shells)

Example 16:

```
 ✘ amit@archlinux  ~/Test  (cd nodir || echo "Redirecting both output and error") > combined.txt 2>&1
 amit@archlinux  ~/Test  cat combined.txt
cd: no such file or directory: nodir
Redirecting both output and error
 amit@archlinux  ~/Test
```

Example 17:

```
 amit@archlinux  ~/Test  (cd nodir || echo "Redirecting both output and error") &> combined.txt
 amit@archlinux  ~/Test  cat combined.txt
cd: no such file or directory: nodir
Redirecting both output and error
 amit@archlinux  ~/Test
```

---

5. Separate Output and Errors:
Redirects `stdout` into one file and `stderr` into another file.

Syntax: command > output_file 2> error_file

Example 18:

```
 amit@archlinux  ~/Test  (cd nodir || echo "This is output") > output.txt 2> error.txt
 amit@archlinux  ~/Test  cat output.txt
This is output
 amit@archlinux  ~/Test  cat error.txt
cd: no such file or directory: nodir
 amit@archlinux  ~/Test
```

---

6. Discard Output and Errors:
Redirects output or errors to /dev/null, effectively discarding it.

```
Syntax: command > /dev/null   (output only)
        command 2> /dev/null  (errors only)
        command  &>  /dev/null     (both errors and output)
```

Example 19:

```
amit@archlinux ~/Test ls
amit@archlinux ~/Test (cd nodir || echo "This is output")  &> /dev/null
amit@archlinux ~/Test ls
amit@archlinux ~/Test
```

---

7. Input Redirection (<):
Input Redirection allows to take input for a command from a file or another source instead of the default standard input (usually the keyboard).

Syntax: command < file

Example 20:

```
amit@archlinux ~/Test sort -M < month.txt

JAN
FEB
MAR
APR
```

---

8. Here Document (<<):
The << operator (Here Document) allows to provide multiline input directly to a command until a specified delimiter is reached.

Syntax: command << delimeter

Example 21:

```
amit@archlinux ~/Test cat << EOF
heredoc> This is multiple line input.
heredoc> This is second line of input.
heredoc> EOF
This is multiple line input.
This is second line of input.
amit@archlinux ~/Test
```

---

8. Here String (<<<):
The <<< operator (Here String) passes a single string as input to a command, making it ideal for quick, one-line inputs.

Syntax: command <<< string

Example 22:

```
amit@archlinux ~/Test cat <<< "hello world"
hello world
```
