ShellInputOutputRedirections

April 15, 2024

1 Shell Input/Output Redirections?

In shell scripting, input/output redirections are used to control where the input of a command comes from and where the output of a command goes. Here are the main types of redirections:

1. **Standard Input (stdin)**: By default, this is the keyboard. The standard input can be redirected from a file using the `<` operator.

   ```
   command < file  # The command takes its input from the file
   ```

2. **Standard Output (stdout)**: By default, this is the terminal. The standard output can be redirected to a file using the `>` operator. If the file already exists, it will be overwritten. If you want to append to the file instead of overwriting it, you can use the `>>` operator.

   ```
   command > file  # The output of the command is written to the file
   command >> file  # The output of the command is appended to the file
   ```

3. **Standard Error (stderr)**: By default, this is the terminal. The standard error can be redirected to a file using the `2>` operator. If the file already exists, it will be overwritten. If you want to append to the file instead of overwriting it, you can use the `2>>` operator.

   ```
   command 2> file  # The error messages from the command are written to the file
   command 2>> file  # The error messages from the command are appended to the file
   ```

4. **Pipes (`|`)**: A pipe takes the output of one command and uses it as the input to another command.

   ```
   command1 | command2  # The output of command1 is used as the input to command2
   ```

5. **Here Documents**: A here document is a type of redirection that allows you to pass multiple lines of input to a command.

   ```
   command << END
   line1
   line2
   line3
   END
   ```

6. **Here Strings**: A here string is a type of redirection that allows you to pass a single line of input to a command.

   ```
   command <<< "input"
   ```

These redirections can be combined in various ways to control the input and output of your shell scripts.

2 Output Redirection?

Output redirection in shell scripting is a feature that allows you to control where the output of a command goes. By default, any output from a command goes to the terminal (standard output). However, you can redirect this output to a file or another command. Here are the main types of output redirection:

1. **Standard Output Redirection (`>` and `>>`)**: The `>` operator redirects the standard output of a command to a file, replacing the current contents of the file. If the file does not exist, it is created. The `>>` operator also redirects the standard output of a command to a file, but it appends the output to the file instead of replacing the current contents.

   ```
   echo "Hello, world!" > file.txt  # Redirects "Hello, world!" to file.txt, replacing its co
   echo "Hello, again!" >> file.txt  # Appends "Hello, again!" to file.txt
   ```

2. **Standard Error Redirection (`2>` and `2>>`)**: The `2>` operator redirects the standard error of a command to a file, replacing the current contents of the file. The `2>>` operator appends the standard error to the file instead of replacing it. Standard error is the output produced by a command when it encounters an error.

   ```
   command 2> error.txt  # Redirects the error output of the command to error.txt
   command 2>> error.txt  # Appends the error output of the command to error.txt
   ```

3. **Redirecting Standard Output and Standard Error to the Same File (`&>` and `&>>`)**: The `&>` operator redirects both the standard output and the standard error of a command to the same file, replacing the current contents of the file. The `&>>` operator appends both the standard output and the standard error to the file.

   ```
   command &> file.txt  # Redirects both the output and error output of the command to file.t
   command &>> file.txt  # Appends both the output and error output of the command to file.tx
   ```

4. **Pipes (`|`)**: The `|` operator redirects the standard output of one command to the standard input of another command. This is useful for chaining commands together.

   ```
   command1 | command2  # The output of command1 is used as the input to command2
   ```

These redirection operators allow you to control where the output of your commands goes, which is a powerful feature of the shell.

3 Input Redirection?

Input redirection in shell scripting is a feature that allows you to control where the input of a command comes from. By default, any input to a command comes from the keyboard (standard input). However, you can redirect this input to come from a file or the output of another command. Here are the main types of input redirection:

1. **Standard Input Redirection (`<`)**: The `<` operator redirects the standard input of a command to come from a file.

```
sort < file.txt  # The sort command takes its input from file.txt
```

2. **Pipes (|)**: The | operator takes the standard output of one command and uses it as the standard input to another command. This is a form of input redirection.

```
cat file.txt | sort  # The output of the cat command is used as the input to the sort comm
```

3. **Here Documents (<<)**: A here document is a type of redirection that allows you to pass multiple lines of input to a command. The << operator is followed by a delimiter, and all lines following the operator up to a line containing only the delimiter are used as the input.

```
cat << END
line1
line2
line3
END
```

4. **Here Strings (<<<)**: A here string is a type of redirection that allows you to pass a single line of input to a command.

```
sort <<< "banana apple cherry"
```

These redirection operators allow you to control where the input of your commands comes from, which is a powerful feature of the shell.

4 Here Document?

A Here Document (also known as a heredoc) is a type of redirection in shell scripting that allows you to pass multiple lines of input to a command. It's particularly useful when you need to provide a large block of text to a command.

The syntax for a Here Document is as follows:

[ ]:
```
command << DELIMITER
text line 1
text line 2
...
DELIMITER
```

In this syntax, `command` is the command you want to run, `DELIMITER` is any string you choose, and the text between the two `DELIMITER` lines is the input to the command. The `DELIMITER` can be any string, but it's often `EOF` (for "end of file") or `END`.

Here's an example of a Here Document with the `cat` command:

[ ]:
```
cat << END
This is a line of text.
This is another line of text.
END
```

In this example, the `cat` command will output the two lines of text.

Here Documents can also be used with variable substitution:

```
[ ]: name="GitHub Copilot"
     cat << END
     Hello, my name is $name.
     I am here to assist you.
     END
```

In this example, `$name` will be replaced with `GitHub Copilot` in the output.

Here Documents are a powerful feature of the shell that allow you to easily provide multi-line input to commands.

5 Discard the output?

In Unix-like operating systems, you can discard the output of a command by redirecting it to the special file `/dev/null`. This file is a type of null device, which discards all data written to it but reports that the write operation succeeded.

Here's how you can use it:

1. **Discard Standard Output**: If you want to discard the standard output of a command, you can use the `>` operator to redirect it to `/dev/null`.

   ```
   command > /dev/null
   ```

   In this example, `command` is the command you want to run. The standard output of `command` will be discarded.

2. **Discard Standard Error**: If you want to discard the standard error of a command, you can use the `2>` operator to redirect it to `/dev/null`.

   ```
   command 2> /dev/null
   ```

   In this example, the standard error of `command` will be discarded.

3. **Discard Both Standard Output and Standard Error**: If you want to discard both the standard output and the standard error of a command, you can use the `&>` operator to redirect them to `/dev/null`.

   ```
   command &> /dev/null
   ```

   In this example, both the standard output and the standard error of `command` will be discarded.

Remember that discarding the output of a command can make it difficult to debug problems, because you won't see any error messages or output. Use this feature wisely.

6 Thank You!