CHAPTER-5 CREATE, VIEW, AND EDIT TEXT FILES

Redirect Output to a File or Program

Standard Input, Standard Output, and Standard Error

A process uses numbered channels called file descriptors to get input and send output. All processes start with at least three file descriptors. Standard input (channel 0) reads input from the keyboard. Standard output (channel 1) sends normal output to the terminal. Standard error(channel 2) sends error messages to the terminal.

A running program, or process, reads input and writes output. When you run a command from the shell prompt, it normally reads its input from the keyboard and sends its output to the terminal window.

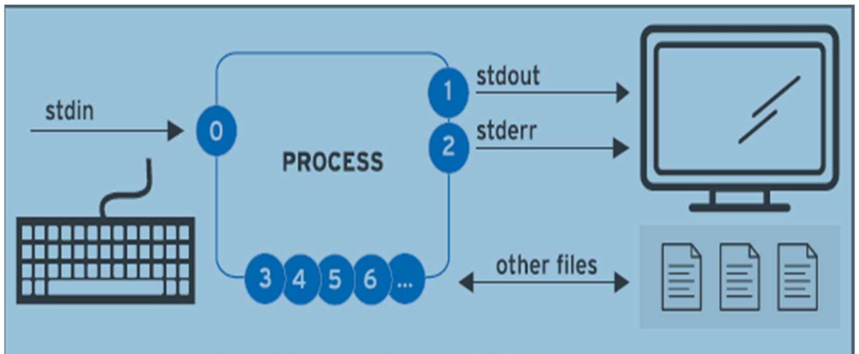
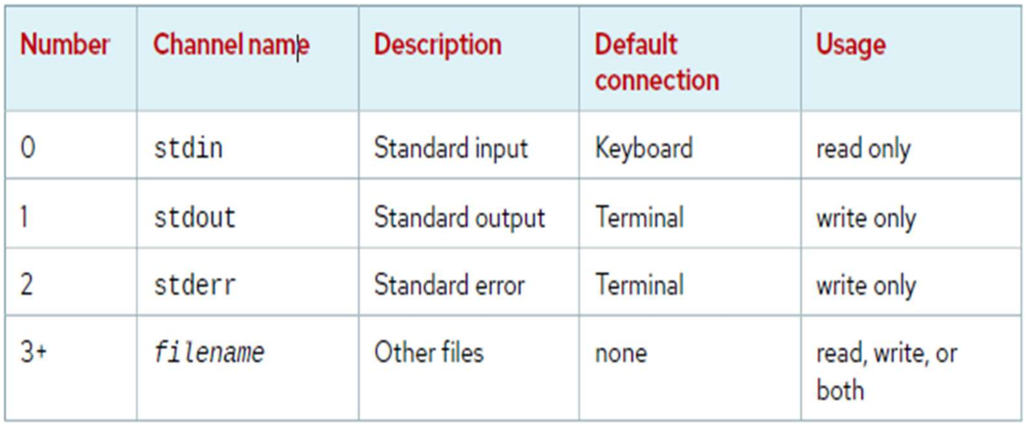


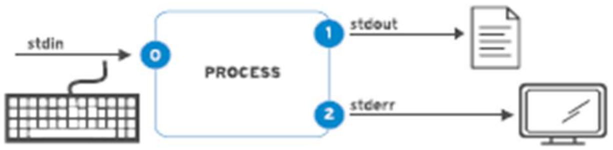
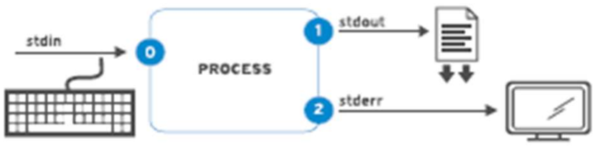
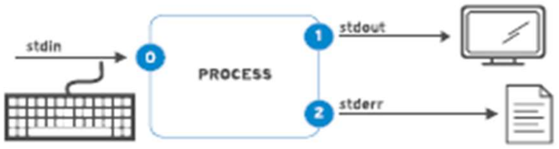
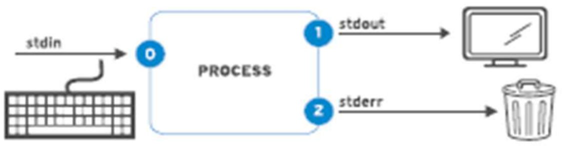
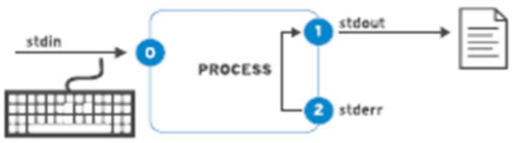
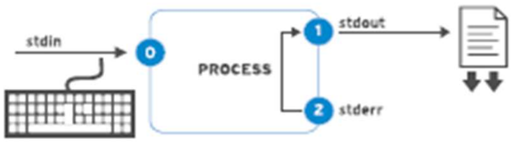
Figure 5.1: Process I/O channels (file descriptors)

Channels (File Descriptors)



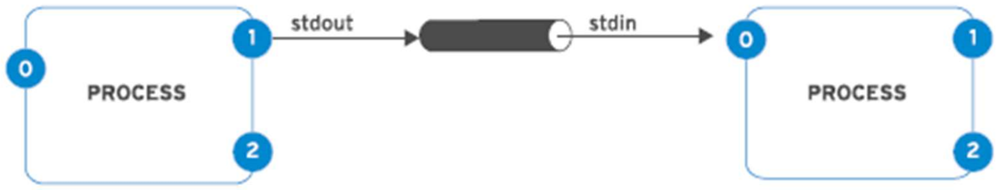
Number	Channel name	Description	Default connection	Usage
0	`stdin`	Standard input	Keyboard	read only
1	`stdout`	Standard output	Terminal	write only
2	`stderr`	Standard error	Terminal	write only
3+	*`filename`*	Other files	none	read, write, or both

Redirect Output to a File

Usage	Explanation	Visual aid
> *file*	Redirect stdout to overwrite a file.	
>> *file*	Redirect stdout to append to a file.	
2> *file*	Redirect stderr to overwrite a file.	
2> /dev/null	Discard stderr error messages by redirecting them to /dev/null.	
> *file* 2>&1 &> *file*	Redirect stdout and stderr to overwrite the same file.	
>> *file* 2>&1 &>> *file*	Redirect stdout and stderr to append to the same file.	

Construct Pipelines

A pipeline is a sequence of one or more commands that are separated by the vertical bar character (|). A pipeline connects the standard output of the first command to the standard input of the next command.



DESCRIPTION	COMMANDS / OPTIONS
Redirect stdout to overwrite a file	**Syntax:** > file **Example:** [user@host ~]$ date > file1.txr ↲
Redirect stdout to append a file	**Syntax:** >>file **Example:** [user@host ~]$ date >> file1.txr ↲
Redirect stderr to overwrite a file	**Syntax:** 2>file **Example:** [user@host ~]$ date 2> file1.txr ↲
Discard stderr error messages	**Syntax:** 2> /dev/null
| Pipeline	Command 1 output | command 2 input **Pipeline Examples:** Redirect the output of the ls command to the less command to display it on the terminal one screen at a time. **Example:** [user@host ~]$ ls -l /usr/bin | less ↲

Edit Text Files from the Shell Prompt

Vim is an improved version of the vi editor, which is distributed with Linux and UNIX systems.
Vim is highly configurable and efficient for practiced users, including split-screen editing, color formatting, and highlighting for editing text.
With the vim-minimal package, you might install the vi editor with core features. This lightweight installation includes only the core features and the basic vi command. You can open a file for editing by using the vi command.

Vim Operating Modes

The Vim editor offers various modes of operation such as command mode, extended command mode, edit mode, and visual mode.

- **Command mode:** When you first open Vim, it starts in command mode, which is used for navigation, cut and paste, and other text modification. Pressing the required keystroke accesses specific editing functions.
- **Insert mode:** An i keystroke enters insert mode, where all typed text becomes file content. Pressing Esc returns to command mode.
- **Visual mode:** A v keystroke enters visual mode, where multiple characters might be selected for text manipulation.

1. Use **Shift+V** for multiline and
2. **Ctrl+V** for block selection.
3. To exit the visual mode, use the **v, Shift+V**, or **Ctrl+V** keystrokes.

Character mode : v
Line mode : Shift+v
Block mode : Ctrl+v

DESCRIPTION	COMMANDS / OPTIONS
Red Hat recommends to learn the following Vim keys and commands.	To create file **Syntax:** vi/vim FileName **Example:** [user@host ~]$ vi hello.txt ↲ **i** Insert mode **u** key undoes the most recent edit. **x** Key deletes a single character. **:w** Command writes (saves) the file and remains in command mode for more editing. **:wq** Command writes (saves) the file and quits Vim. **:q!** Command quits Vim, and discards all file changes since the last write.

Vim Configuration Files

The /etc/vimrc and ~/.vimrc configuration files alter the behavior of the vim editor for the entire system or a specific user respectively.

DESCRIPTION	COMMANDS / OPTIONS
Edit	**a** Insert text after current cursor location **SHIFT+A** Insert text at the end of current line **SHIFT+x** Deletes the character before the cursor location **dd** Deletes the line the cursor is on **yy** Copies the current line **p** Paste
Navigation	**k** Moves the cursor up one line **j** Moves the cursor down one line **h** Moves the cursor to the left one-character position **l** Moves the cursor to the right one-character position **0** Positions cursor at beginning of line **$** Positions cursor at end of line SIFT+H Move to top of screen **SIFT+L** Move to bottom of screen

DESCRIPTION	COMMANDS / OPTIONS
Search and Replace	**To search string in vi** **Syntax:** :s/string **Example:-** :s/wordToSearch ↵ **To replacing one string with another string** **Syntax:** :s/pattern/replace/ **Example:-** :s/OneString/Another ↵ **To replacing every occurrence of a string in the entire text is similar** **Syntax:** :%s/pattern/replace/ **Example:-** :%s/OneString/Another ↵ **?string** search backward for occurrence of string in text **n** move to next occurrence of search string **N** move to next occurrence of search string in opposite direction

Change the Shell Environment

Set shell variables to run commands, and edit Bash startup scripts to set shell and environment variables to modify the behavior of the shell and programs that are run from the shell.

Shell Variable Usage

You can set shell variables to help to run commands or to modify the behavior of the shell. You can also export shell variables as environment variables, which are automatically copied to programs that are run from that shell. You can use variables for ease of running a command with a long argument, or to apply a common setting to commands that are run from that shell.

Assign Values to Variables

Assign a value to a shell variable with the following syntax:
[user@host ~]$ **VARIABLENAME=value** ↵

Variable names can contain uppercase or lowercase letters, digits, and the underscore character
Example:-

```
[user@host ~]$ first_name=Harry ↵
[user@host ~]$ COUNT=40   ↵
```

This change affects only the shell that you run the command in, not any other shells that you might be running on that server. You can **use the set command to list all shell variables that are currently set**. (It also lists all shell functions, which you can ignore.) To improve readability, you can pipe the output to the less command so that you can view it one page at a time.

Retrieve Values with Variable Expansion

You can use variable expansion to refer to the value of a variable that you set. To use variable expansion, precede the name of the variable with a dollar sign ($). In the following examples, the variable expansion occurs first and then the echo command prints the rest of the command line that is entered.

Example:-

To sets the variable COUNT to 40.

```
[user@host ~]$ COUNT=40 ↵
```

To prints the COUNT variable value **echo** command use

```
[user@host ~]$ echo $COUNT ↵
```

Configure Bash with Shell Variables

Some shell variables are set when Bash starts. You can modify them to adjust the shell's behavior.
Example, the HISTFILE, HISTFILESIZE, and HISTTIMEFORMAT

Configure Programs with Environment Variables

The shell provides an environment to the programs that you run from that shell. You can assign any variable that is defined in the shell as an environment variable by marking it for export with **the export** command.

```
[user@host ~]$ EDITOR=vim ↵
[user@host ~]$ export EDITOR ↵
```

You can set and export a variable in one step:

```
[user@host ~]$ export EDITOR=vim ↵
```

Another important environment variable is PATH. The **PATH** variable contains a list of colon-separated directories that contain programs:

```
[user@host ~]$ echo $PATH ↵
```

You can append the /home/user/sbin directory to your PATH for the current session as follows:

```
[user@host ~]$ export PATH=${PATH}:/home/user/sbin ↵
```

To list all the environment variables for a shell, run the env command:

```
[user@host ~]$ env
```

The EDITOR environment variable specifies your default text editor for command-line programs.

```
[user@host ~]$ export EDITOR=nano ↲
```

Set Variables Automatically

When Bash starts, several text files run with shell commands that initialize the shell environment.
To set shell or environment variables automatically when your shell starts, you can edit these Bash startup scripts.

The **/etc/profile** and ~/.bash_profile files configure the Bash environment.
The **/etc/profile** and **~/.bash_profile** files also source the **/etc/bashrc** and **~/.bashrc** files respectively.

For interactive non-login shells, only the **/etc/bashrc** and **~/.bashrc** files configure the Bash environment.
While the **/etc/profile** and **/etc/bashrc** files apply to the whole system, the **~/.bash_profile** and **~/.bashrc** files are user-specific.

Non-interactive shells invoke any files that the BASH_ENV variable defines. This variable is not defined by default.
To create a variable that is available to all of your interactive shells, edit the **~/.bashrc** file. To apply a variable only once after the user logs in, define it in the **~/.bash_profile** file.
Bash Aliases: Bash aliases are shortcuts to other Bash commands.
Example, if you must frequently type a long command, then you can create a shorter **alias** to invoke it.

```
[user@host ~]$ alias hello='echo "Hello, this is a long string."' ↲
```

You can then run the hello command and it invokes the echo command.

```
[user@host ~]$ hello ↲
```

Unset and Unexport Variables and Aliases

To unset and unexport a variable, use the unset command:

```
 [user@host ~]$ unset file1 ↲
```

To unexport a variable without unsetting it, use the **export -n** command:

```
[user@host ~]$ export -n PS1 ↲
```

To unset an alias, use the unalias command:

```
[user@host ~]$ unalias hello ↲
```