Creating, Viewing and Editing Text Files from the command output or in a text editor.

Objectives:

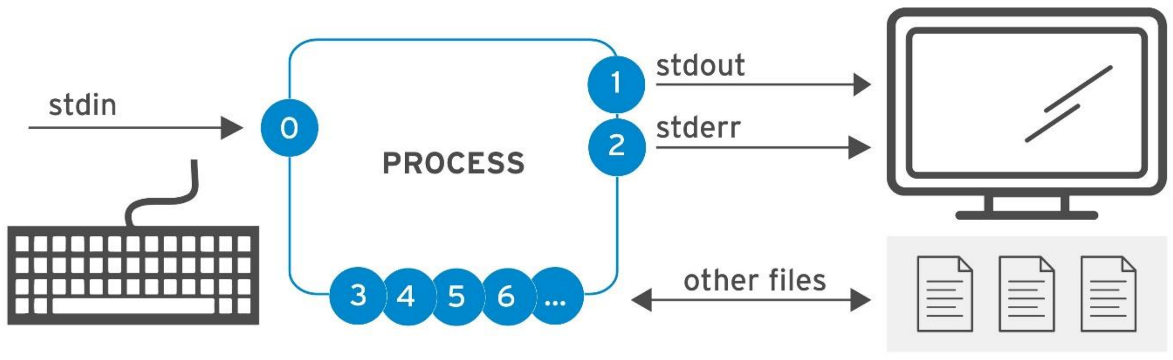
1. Save command output or errors to a file with shell redirection, and process command output through multiple command-line programs with pipes.
2. Create and edit text files using the ***vim*** editor.
3. Use shell variables to help run commands, and edit Bash startup scripts to set shell and environment variables to modify the behavior of the shell and programs run from the shell.

REDIRECTING OUTPUT TO A FILE OR PROGRAM

1. STANDARD INPUT, STANDARD OUTPUT AND STANDARD ERROR

A running program, or process, needs to read input from somewhere and write output to somewhere. A command run from the shell prompt normally reads its input from the keyboard and sends its output to its terminal window.

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. If a program opens separate connections to other files, it may use higher-numbered file descriptors.



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 and/or write

2. REDIRECTING OUTPUT TO A FILE

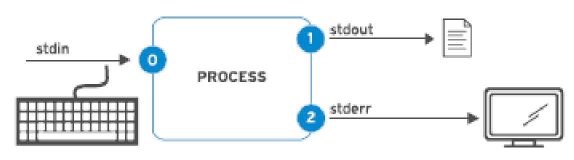
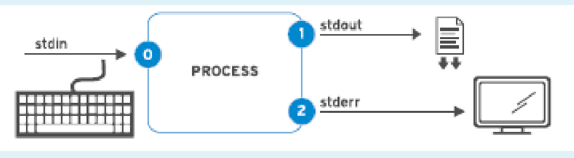
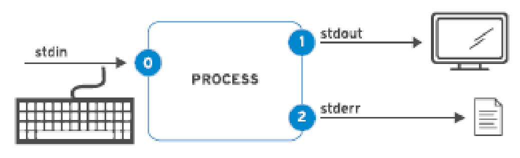
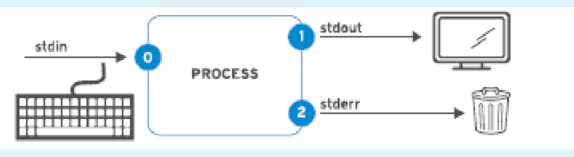
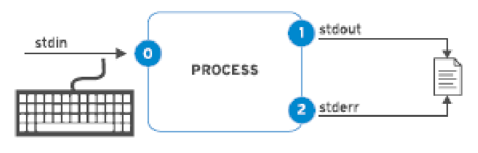
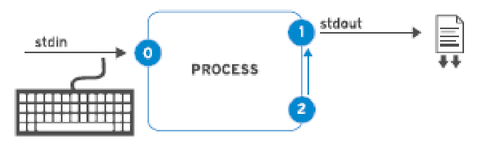
I/O redirection changes how the process gets its input and output. Instead of getting input from the keyboard, or sending output and errors to the terminal, the process reads from or writes to files.

Redirection lets you save messages to a file that are normally sent to the terminal window. Alternatively, you can use redirection to discard output or errors, so they are not displayed on the terminal or saved.

Redirecting ***stdout*** suppresses process output from appearing on the terminal. As seen in the following table, redirecting only ***stdout*** does not suppress ***stderr*** error messages from displaying on the terminal. If the file does not exist, it will be created. If the file does exist and the redirection is not one that appends to the file, the file's contents will be overwritten.

If you want to discard messages, the special file **/dev/null** quietly discards channel output redirected to it and is always an empty file.

Output Redirection Operators

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 to **`/dev/null`**	
> *file* 2>&1	redirect **`stdout`** and **`stderr`** to overwrite the same file	
&> *file*		
>> *file* 2>&1	redirect **`stdout`** and **`stderr`** to append to the same file	
&>> *file*		

Examples for Output Redirection

1) $date > file1

   Save a time stamp in file1

2) $tail –n 10 file1 > file2

   Copy the last 10 lines from file1 to file2

3) $cat file1 file2 file3 file4 > file5

   Concatenate four files into one

4) $echo “new line of information” >> file1

   Append output to an existing file

5) $find /etc –name passwd 2> file4

   Redirects errors to a file while viewing normal command output on the terminal

6) $find /etc –name passwd > /tmp/output 2> /tmp/errors

   Save process output and error messages to separate files

7) $find /etc –name passwd &> /tmp/save-both

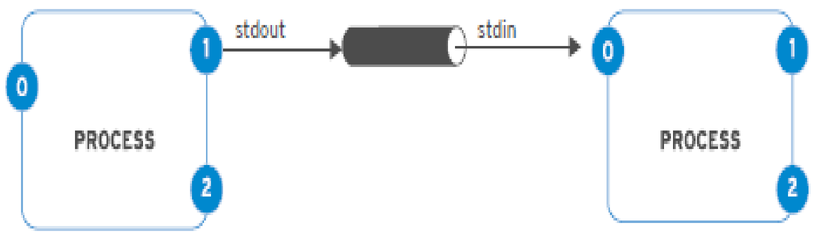
    Store output and generated errors to an existing file

8) $find /etc –name passwd >> /tmp/save-both 2>&1

    Append output and generated errors to an existing file

3. CONSTRUCTING PIPELINES

A *pipeline* is a sequence of one or more commands separated by the *pipe* character (). A pipe connects the standard output of the first command to the standard input of the next command.



Process I/O Mapping

Pipelines allow output of a process to be manipulated and formatted by other processes before it is output to the terminal. One useful mental image is to imagine that data is “flowing” through the pipeline from one process to another, being altered slightly by each command in the pipeline through which it flows.

Pipeline Examples

1) $ls -l /usr/bin | more -10

    The output of **ls** command to **more** command to display 10 lines on the terminal

2) $ls | wc -10

    The output of the **ls** command is piped to **wc –l** , which counts the number of lines received from **ls** and prints that to the terminal

3) $ls -t | head -n 10 > /tmp/ten-last-changed –files

    In this pipeline, **head** will output the first 10 lines of output from **ls -t,** with the final result redirected to a file.

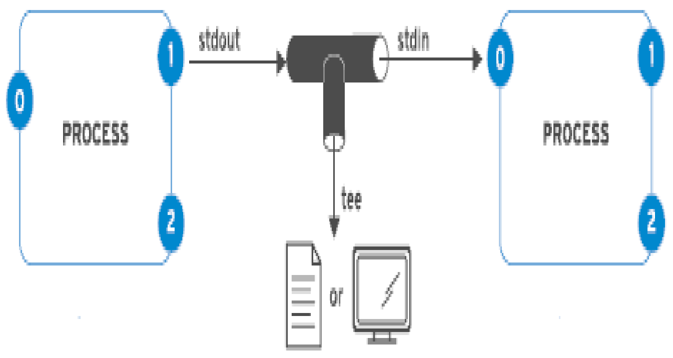
Pipelines, Redirection, and the tee Command

When redirection is combined with a pipeline, the shell sets up the entire pipeline first, then it redirects input/output. If output redirection is used in the *middle* of a pipeline, the output will go to the file and not to the next command in the pipeline.

In this example, the output of the **ls** command goes to the file, and **less** displays nothings on the terminal

```
$ls > /tmp/saved-output | less
```

The **tee** command overcomes this limitation. In a pipeline, **tee** copies its standard input to its standard output and also redirects its standard output to the files named as arguments to the command.



Process I/O Mapping with tee

Pipeline Examples Using with the tee Command

This example redirects the output of the **ls** command to the file and passes it to **less** to be displayed on the terminal one scree at a time.

```
$ls -l | tee /tmp/saved-output | less
```

If **tee** is used at the end of a pipeline, then the final output of a command can be saved and output to the terminal at the same time.

```
$ls -t | head -n 10 | tee /tmp/ten-last-changed-files
```

EDITING TEXT FILES FROM THE SHELL PROMPT

1. EDITING FILES WITH VIM

A key design principle of Linux is that information and configuration settings are commonly stored in text-based files. These files can be structured in various ways, as lists of settings, in INI-like formats, as structured XML or YAML, and so on. However, the advantage of text files is that they can be viewed and edited using any simple text editor.

Vim is an improved version of the vi editor distributed with Linux and UNIX systems. Vim is highly configurable and efficient for practiced users, including such features as split screen editing, color formatting, and highlighting for editing text.

Why Learn Vim?

You should know how to use at least one text editor that can be used from a text-only shell prompt. If you do, you can edit text-based configuration files from a terminal window, or from remote logins through ssh or the Web Console. Then you do not need access to a graphical desktop in order to edit files on a server, and in fact that server might not need to run a graphical desktop environment at all.

But then, why learn Vim instead of other possible options? The key reason is that Vim is almost always installed on a server, if any text editor is present. This is because vi was specified by the POSIX standard that Linux and many other UNIX-like operating systems comply with in large part.

In addition, Vim is often used as the vi implementation on other common operating systems or distributions. For example, macOS currently includes a lightweight installation of Vim by default. So Vim skills learned for Linux might also help you get things done elsewhere.

Starting Vim

Vim may be installed in Red Hat Enterprise Linux in two different ways. This can affect the features and Vim commands available to you.

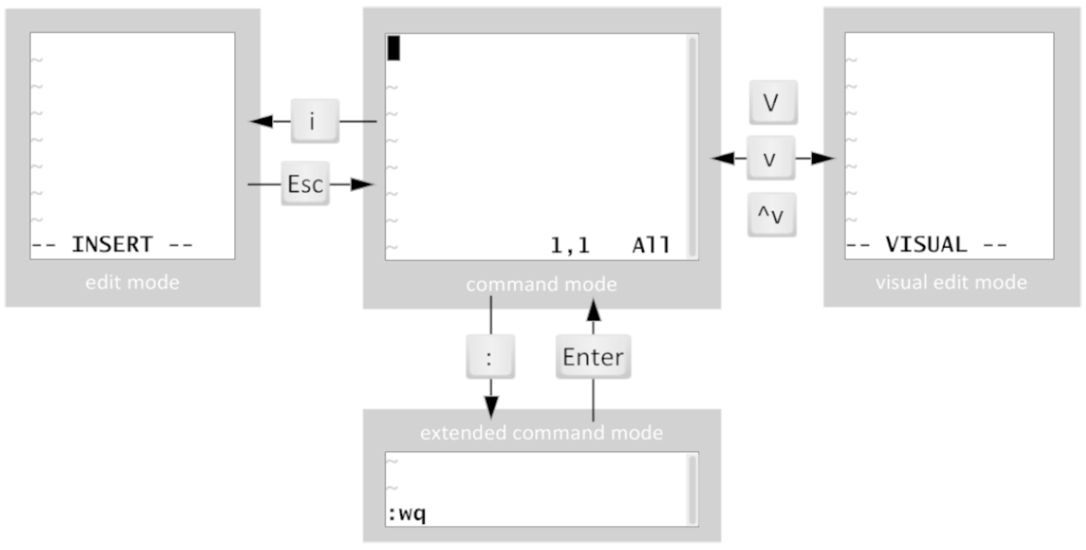
Your server might only have the vim-minimal package installed. This is a very lightweight installation that includes only the core feature set and the basic vi command. In this case, you can open a file for editing with vi filename, and all the core features discussed in this section will be available to you.

Alternatively, your server might have the vim-enhanced package installed. This provides a much more comprehensive set of features, an on-line help system, and a tutorial program. In order to start Vim in this enhanced mode, you use the vim command.

```
[user@localhost ~] vim filename
```

Vim Operating Modes

An unusual characteristic of Vim is that it has several modes of operation, including command mode, extended command mode, edit mode, and visual mode. Depending on the mode, you may be issuing commands, editing text, or working with blocks of text. As a new Vim user, you should always be aware of your current mode as keystrokes have different effects in different modes.



Moving between Vim modes

When you first open Vim, it starts in command mode, which is used for navigation, cut and paste, and other text manipulation. Enter each of the other modes with single character keystrokes to access specific editing functionality:

- An **i** keystroke enters *insert* mode, where all text typed becomes file content. Pressing **Esc** returns to command mode.
- A **v** keystroke enters *visual* mode, where multiple characters may be selected for text manipulation. Use **Shift+V** for multiline and **Ctrl+V** for block selection. The same keystroke used to enter visual mode (v, Shift+V or Ctrl+V) is used to exit.
- The **:** keystroke begins extended command mode for tasks such as writing the file (to save it), and quitting the Vim editor.

The Minimum, Basic Vim Workflow

The **i** key puts Vim into insert mode. All text entered after this is treated as file contents until you exit insert mode. The **Esc** key exits insert mode and returns Vim to command mode. The **u** key will undo the most recent edit. Press the **x** key to delete a single character. The **:w** command writes (saves) the file and remains in command mode for more editing. The **:wq** command writes (saves) the file and quits Vim. The **:q!** command quits Vim, discarding all file changes since the last write. The Vim user must learn these commands to accomplish any editing task.

Rearranging Existing Text

In Vim, copy and paste is known as **yank** and **put**, using command characters **Y** and **P**. Begin by positioning the cursor on the first character to be selected, and then enter visual mode. Use the arrow keys to expand the visual selection. When ready, press **y** to **yank** the selection into memory. Position the cursor at the new location, and then press **p** to **put** the selection at the cursor.

Visual Mode in Vim

Visual mode is a great way to highlight and manipulate text. There are three keystrokes:

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

Character mode highlights sentences in a block of text. The word **VISUAL** will appear at the bottom of the screen. Press **v** to enter visual character mode. **Shift+V** enters line mode. **VISUAL LINE** will appear at the bottom of the screen.

Visual block mode is perfect for manipulating data files. From the cursor, press the **Ctrl+V** to enter visual block. **VISUAL BLOCK** will appear at the bottom of the screen. Use the arrow keys to highlight the section to change.

Practice Tasks

1. Open vimtutor
2. In the **vimtutor** window, perform Lesson 1.1
3. In the **vimtutor** window, perform Lesson 1.2
4. In the **vimtutor** window, perform Lesson 1.3

   Vim has fast, efficient keystrokes to delete an exact amount of words, lines, sentences, and paragraphs. However, any editing job can be accomplished using x for single character deletion.
5. In the **vimtutor** window, perform Lesson 1.4

   For most editing tasks, the first key pressed is **i**
6. In the **vimtutor** window, perform Lesson 1.5

   In the lecture, only the i (insert) command was taught as the keystroke to enter edit mode. This vimtutor lesson demonstrates other available keystrokes to change the cursor placement when insert mode is entered. In insert mode, all typed text is file content.
7. In the **vimtutor** window, perform Lesson 1.6

   Type :wq to save the file and quit the editor.