VIM Excercises

Legend:

Input Command

Output of the previous command

Prerequisites

- Ubuntu 22.04 Server Powered up
- Ubuntu Server on Bridged mode
- From Host OS, ssh to the Ubuntu Server

Show your work

- Screenshots of the cli commands
- Screenshots of the outputs

Fundamental Linux knowledge on VIM

Vim (Vi IMproved) is a highly configurable text editor built to enable efficient text editing. It's an improvement over the original Vi editor distributed with UNIX systems, providing a more complete feature set while still retaining the efficiency and modal editing features that set Vi apart from other editors. Vim is ubiquitous in the Linux world, often used for programming, system administration, and as a general-purpose text editor.

Vim's prominence in the Linux ecosystem and its utility in secure or restricted environments, such as classified or air-gapped systems, can be attributed to several factors that make it not just beneficial but essential to master for those working in such contexts.

Vim's Native Presence in Linux

Vim, being a direct descendant of the Vi editor, is available on virtually all Unix-like systems, including Linux. This ubiquity stems from the Unix philosophy of creating small, modular tools that do their jobs well. Vim is often pre-installed on Linux systems, making it a readily available tool for editing configuration files, writing scripts, or any form of text editing directly in the command line environment.

Example: Editing Configuration Files

Consider a scenario where you need to edit the `/etc/ssh/sshd_config` file on a Linux server to change its SSH configuration. With Vim, you can simply execute `sudo vim`

`/etc/ssh/sshd_config` to open the file in an editor that's fully featured yet operates within a terminal, requiring no graphical interface.

Essential for Working on Classified or Air-Gapped Systems

Classified or air-gapped systems are isolated from the internet to prevent unauthorized access and ensure the security of sensitive information. The use of software on such systems is highly controlled, and administrators often rely on tools that are already available in the system's basic installation.

Vim's Minimal Dependency Footprint

Vim's minimal dependency footprint makes it ideal for secure environments. It doesn't require additional libraries or frameworks to function, minimizing the risk of vulnerabilities.

Example: Secure Script Editing

Imagine you're developing a script to automate data processing on an air-gapped classified system. Using Vim, you can comfortably write and test your script (`process_data.sh`) in an environment that doesn't allow for the installation of additional software. Vim's syntax highlighting, error detection, and auto-completion features (enabled through Vim plugins or configurations) support efficient script development.

Vim's Role in System Administration and Scripting

System administrators often need to work directly on servers through secure shell (SSH) connections, where graphical text editors are not an option. Vim's powerful text manipulation capabilities make it an excellent tool for this purpose.

Example: Secure Configuration and Scripting

In managing an air-gapped server, tasks like updating user permissions, scheduling cron jobs, or tweaking network settings require direct editing of system files such as `/etc/passwd`, `crontab`, or `/etc/network/interfaces`. Vim allows administrators to perform these tasks efficiently through command-line interfaces.

Learning Vim for Secure Environments

Given Vim's prevalence in Linux and its utility in restricted environments, understanding its basics to advanced features becomes a valuable skill for anyone working in cybersecurity, system administration, and development roles within such contexts. Mastery of Vim not only ensures efficiency but also compliance with the security protocols that restrict the use of unauthorized external software.

Security-Conscious Editing

Vim supports encrypted file editing, which is critical in handling sensitive data. For example, you can use `:X` command in Vim to apply encryption to a file, prompting you for a passphrase. This feature is particularly useful when working with confidential data files on classified systems, ensuring that data remains secure even if it's inadvertently moved or accessed.

Why you should care?

Vim's integration with Linux, combined with its efficiency and security features, makes it an indispensable tool for professionals working on classified or air-gapped systems. Its ability to operate within the constraints of highly secure environments, where minimal external dependencies and a high degree of control and security are required, underscores the importance of learning Vim for anyone involved in these areas.

VIM Key Features

- **Modal Editing:** Vim operates in several modes, primarily normal, insert, and visual modes, each serving different purposes. This allows for efficient text manipulation and navigation.

Vim's efficiency as a text editor largely stems from its mode-based editing system, which allows users to perform a wide array of text manipulation tasks efficiently without taking their hands off the keyboard. This design philosophy facilitates a workflow where actions are composed, allowing for complex edits to be made with a few keystrokes. Let's delve deeper into the specifics of Normal, Insert, and Visual modes, which are foundational to understanding and mastering Vim.

VIM MODES

Normal Mode

Normal Mode is the default state of Vim when you open a document. It's designed for navigating and manipulating text rather than inserting text. This mode turns the keyboard into a powerful tool for editing commands.

- **Navigation:** Use `h` (left), `j` (down), `k` (up), and `l` (right) for moving the cursor around the text efficiently. You can also use commands like `w` to jump forward a word, `b` to go back a word, and `0` or `$` to jump to the start or end of a line, respectively.
- **Editing:** Delete characters, words, or lines with commands like `x` (delete character), `dw` (delete word), and `dd` (delete line). These commands can be prefixed with numbers for repeating actions, such as `2dd` to delete two lines.

- **Copying and Pasting:** Use `y` to yank (copy) text and `p` to paste it. Similar to deleting, you can yank words with `yw` or whole lines with `yy`. The pasted content will be inserted after the cursor position or in the line below.
- **Undo and Redo:** Press `u` to undo the last action and `Ctrl+r` to redo.

The power of Normal Mode lies in its ability to perform complex text manipulations quickly, with minimal keystrokes, making routine editing tasks far more efficient.

Insert Mode

Insert Mode is where you can type text into your document naturally. Unlike Normal Mode, where each key performs a specific command, keys in Insert Mode input text as you would expect in a conventional text editor.

- **Entering Insert Mode:** From Normal Mode, press `i` to start inserting text at the cursor's current position. Other variations include `a` (append) to start inserting after the cursor, `o` (open) to insert a new line below the current line, and their uppercase counterparts `I`, `A`, `O` for inserting at the beginning of the line, appending at the end of the line, and opening a new line above the current line, respectively.
- **Exiting Insert Mode:** Press `Esc` to return to Normal Mode. This transition is crucial for leveraging Vim's powerful editing capabilities, as you'll often switch between inserting text and using Normal Mode commands to manipulate text.

Insert Mode focuses on straightforward text input, making it similar to typing in more traditional text editors, but with the ability to quickly switch back to Normal Mode for editing.

Visual Mode

Visual Mode is used for selecting text. Once text is selected, you can perform operations on it, such as deleting, copying, or applying custom commands. Visual Mode provides a way to specify exactly what part of the text you want to manipulate.

- **Entering Visual Mode:** Press `v` in Normal Mode to start selecting text character by character. Use `V` for line selection, which selects entire lines at a time. For block selection (columnar selection), press `Ctrl+v`. This mode is especially useful for editing configurations or code where you need to select and manipulate blocks of text.
- **Modifying Selection:** Once in Visual Mode, you can use the same navigation keys as in Normal Mode to expand or reduce your selection.
- **Performing Actions:** After selecting text, you can apply any command you would in Normal Mode, such as `d` to delete the selection or `y` to yank it. Pressing `p` to paste will replace the selected text with the contents of the clipboard.

Visual Mode bridges the gap between the character-based editing of Normal Mode and the need to manipulate larger chunks of text efficiently, allowing for precise text selection and manipulation.

SUMMARY

Understanding and becoming proficient in these three modes forms the foundation of effective Vim usage. Each mode serves distinct purposes: Normal Mode for navigating and manipulating text, Insert Mode for typing and entering text, and Visual Mode for selecting and acting on specific text regions. Mastering the transitions and commands within these modes unlocks the true potential of Vim, making it an incredibly powerful tool for editing text at high speed and with great precision.

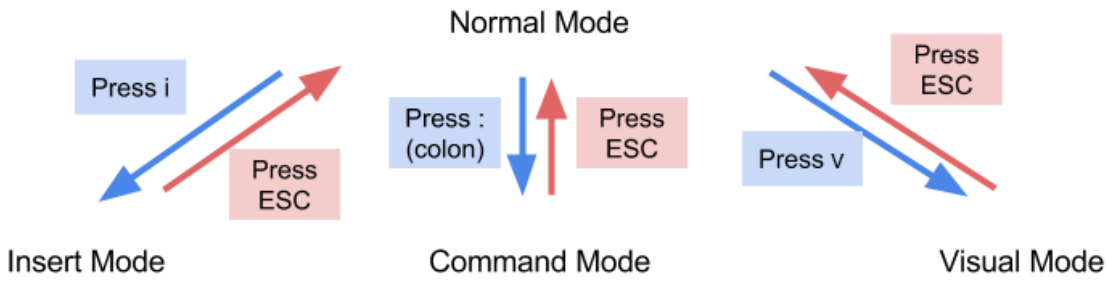
more VIM Key Features

- **Extensibility:** Users can customize Vim with plugins, scripts, and by editing the `.vimrc` configuration file to tailor the environment to their liking.
- **Powerful Search and Replace:** Vim offers robust search and replace capabilities, including regular expressions.
- **Syntax Highlighting:** Supports syntax highlighting for a wide range of programming languages, making it easier to read code.
- **Split Screen:** Allows working with multiple files simultaneously by splitting the window vertically or horizontally.
- **Buffers, Windows, and Tabs:** Vim manages files using buffers. It can display buffers in windows, and you can organize windows in tabs for a comprehensive overview of multiple files.
- **Command-Line Integration:** Vim can execute shell commands without leaving the editor, and you can even filter text through shell commands.

Operating Modes

Understanding Vim's modes is crucial to using Vim effectively:

1. **Normal Mode:** The default mode where you can execute commands. This mode doesn't allow direct text insertion but is powerful for navigating and manipulating text.
2. **Insert Mode:** Entered from normal mode by pressing `i` (insert), `a` (append), or other similar commands. In insert mode, you can type text normally.
3. **Visual Mode:** Entered from normal mode by pressing `v` (character selection), `V` (line selection), or `Ctrl+v` (block selection). Visual mode is used for selecting blocks of text.
4. **Command-Line Mode:** Accessed from normal mode by pressing `:`. In this mode, you can enter commands to save files, search, perform global substitutions, and more.



Switching MODES in vim

Vim, with its mode-based editing philosophy, offers a powerful and efficient way to work with text by allowing the user to switch between different modes, each optimized for specific tasks.

Understanding how to switch between these modes is fundamental to mastering Vim. The primary modes are Normal Mode, Insert Mode, Visual Mode, Command-Line Mode, and others like Replace Mode.

Normal Mode to Other Modes

- **Normal Mode** is the default mode when you open Vim. It's where you can execute Vim commands to manipulate text without inserting or deleting text directly.
    - **To Insert Mode:** Press `i` to start inserting text at the cursor, `I` to insert at the beginning of the line, `a` to append after the cursor, `A` to append at the end of the line, `o` to open a new line below the current line, or `O` to open a new line above it. These commands switch Vim to Insert Mode, where you can type text normally.
    - **To Visual Mode:** Press `v` to start character-wise visual selection, `V` for line-wise selection, or `Ctrl+v` for block-wise selection. In Visual Mode, you can select text for manipulation.
    - **To Command-Line Mode:** Press `:` to enter Command-Line Mode at the bottom of the Vim window, where you can enter commands such as `:w` to save or `:q` to quit.
    - **To Replace Mode:** Press `R` to replace the character under the cursor and continue replacing text as you type, without automatically moving to the next line after reaching the end.

Insert Mode to Normal Mode

- **From Insert Mode:** Press `Esc` (Escape key) to return to Normal Mode. This is the key to remember since Insert Mode is where you'll likely spend a lot of time typing text, and `Esc` brings you back to Normal Mode, where you can execute commands.

Visual Mode to Normal Mode

- **From Visual Mode:** Press `Esc` to exit Visual Mode and return to Normal Mode. In Visual Mode, after you've selected text, you might copy (`y`), cut (`d`), or change (`c`) the selection, and then press `Esc` to return to Normal Mode.

Command-Line Mode to Normal Mode

- **From Command-Line Mode:** After entering a command, pressing Enter will execute the command. Vim automatically returns to Normal Mode after the command is executed. If you're in the command-line window (after pressing `q:` for example) and haven't executed a command, pressing `Esc` will cancel the command-line window and return you to Normal Mode.

Insert Mode to Command-Line Mode and Visual Mode

Direct switching from Insert Mode to Command-Line Mode or Visual Mode is not typical. You usually go through Normal Mode as an intermediate step. For instance, to go from Insert Mode to Command-Line Mode, you would press `Esc` to enter Normal Mode, then `:` to enter Command-Line Mode. Similarly, to go from Insert Mode to Visual Mode, you press `Esc` to return to Normal Mode first, then enter the desired Visual Mode command (`v`, `V`, or `Ctrl+v`).

Summary

Switching modes in Vim is primarily about knowing the key(s) to press to enter your desired mode from the current one. The `Esc` key is pivotal, as it usually returns you to Normal Mode, where you can then branch out to other modes. Mastering these transitions is key to fluidly moving through your text manipulation tasks in Vim, enhancing your efficiency and productivity.

Navigation and Commands - NORMAL MODE

Vim commands can seem obscure at first, but they allow for powerful and precise control:

- **Navigation:** Use `h`, `j`, `k`, `l` for left, down, up, right movement. There are also commands for jumping to the start/end of a line, next word, etc.
- **Editing:** Commands exist for deleting (`d`), copying (`y`), pasting (`p`), undoing (`u`), and more, with the ability to combine them with navigation commands for quick edits.
- **Customization:** The `.vimrc` file in your home directory allows you to customize Vim extensively, from remapping keys to setting preferences like auto-indenting or enabling line numbers.

Advanced Features

Vim also supports more advanced features like:

- **Macro recording:** Automate repetitive tasks by recording keystrokes and replaying them.
- **Code folding:** Collapse sections of code for better readability.
- **Integration with external tools:** Filter text through external commands, integrate with version control systems, etc.

Learning Curve

Vim has a reputation for having a steep learning curve, primarily due to its unique modal editing approach and the vast array of commands. However, many users find that the investment in learning Vim pays off in increased productivity and efficiency.

To get started with Vim, you can run the `vimtutor` command in your terminal, which provides a hands-on introduction to the basics of Vim's commands and modes. There are also numerous resources, tutorials, and communities dedicated to Vim where beginners can find support and tips.

EXERCISES

SIMPLE EXERCISES

Exercise 1: Open a File in Vim

Objective: Open an existing file named `example.txt` in the current directory using Vim.
Instructions:

1. Navigate to the directory containing `example.txt`.
2. Use the Vim command to open the file by typing `vim example.txt`.
3. Once open, navigate through the file using the arrow keys.
4. Exit Vim without making changes by pressing `:q!` and then Enter.

Exercise 2: Editing and Saving Changes

Objective: Edit `document.txt` by adding a new line of text and save the changes. **Instructions:**

1. Open `document.txt` with Vim by typing `vim document.txt`.
2. Enter insert mode by pressing `i`.
3. Move to the end of the file and add the following line: `This is a new line of text`.
4. Save the changes and exit Vim by pressing `Esc` to leave insert mode, then type `:wq` and press Enter.

Exercise 3: Search for Text within a File

Objective: Open `report.txt`, search for the word "summary", and navigate to the first occurrence. **Instructions:**

1. Open `report.txt` in Vim by typing `vim report.txt`.
2. To search for "summary", type :`/summary` and press Enter.
3. Once the first occurrence is highlighted, you can press `n` to move to the next occurrence or `N` to move to the previous one.
4. Exit Vim by typing `:q!` and press Enter.

INTERMEDIATE EXERCISES

wget in linux explained

`wget` is a highly versatile command-line utility in Linux designed for downloading files from the web. It stands for "World Wide Web get." Originally created by Hikaru Yoshimura and currently maintained by Tim Rühsen, it supports downloading via HTTP, HTTPS, and FTP protocols, which covers the vast majority of file transfer scenarios on the internet.

Features and Capabilities

- **Robust File Retrieval:** `wget` can download files in the background, resume downloads that were interrupted due to network problems or system crashes, and handle large file downloads efficiently.
- **Recursive Downloading:** One of `wget`'s most powerful features is its ability to recursively download websites, meaning it can mirror entire websites by downloading web pages, images, and files linked from those pages.
- **Non-interactive Usage:** `wget` is non-interactive, which allows it to run in the background while the user is not logged on. This makes it ideal for scripts and cron jobs to automate download tasks.
- **Bandwidth Control:** Users can limit the download speed to avoid using too much bandwidth during file transfers.
- **HTTP Features:** `wget` supports HTTP cookies, proxy servers, and headers. This allows it to perform downloads that might require authentication or custom HTTP requests.

Basic Usage

The basic syntax of `wget` is:

```
wget [options] [URL]
```

For example, to download a file from a given URL, you would use:

```
wget https://example.com/file.zip
```

This command would download the file `file.zip` from the specified URL and save it in the current directory.

Recursive Downloading: To recursively download a website, you can use the `-r` (or `--recursive`) option:

```
wget -r https://example.com
```

Specifying Download Directory: To download files to a specific directory, use the `-P` option:

```
wget -P /path/to/directory https://example.com/file.zip
```

Specifying using IPv4 only: To enforce `wget` to use only IPv4 when downloading, you can use the `--inet4-only` option. This option prevents `wget` from attempting to connect via IPv6, ensuring that connections are made through IPv4 only.

Here's how you would use this option in a command:

```
wget --inet4-only http://example.com/file.zip
```

This command instructs `wget` to download `file.zip` from `http://example.com` using only IPv4.

Additional Context

- **Why Specify IPv4?** There might be scenarios where IPv6 connectivity is problematic or unsupported by the server hosting the file, leading to failed download attempts. In such cases, forcing `wget` to use IPv4 can resolve the issue.
- **Compatibility:** The `--inet4-only` option is supported by most versions of `wget`. However, if you encounter a version where it's not recognized, consulting the `wget` man page (`man wget`) or help output (`wget --help`) can provide alternatives or confirm the option's availability.
- **IPv6 Equivalent:** Conversely, if you ever need to enforce IPv6 usage, `wget` provides the `--inet6-only` option for this purpose.

This feature of `wget` is particularly useful for scripting and automation in environments where IPv4 is preferred or required for connectivity.

SUMMARY

`wget` is a powerful and flexible tool that is essential for users who frequently work with file downloads in the Linux command line. Its wide range of features from simple file downloads to complex site mirroring, combined with its ability to work in scripts and automate tasks, make it an indispensable tool for system administrators, developers, and data analysts alike.

What is the Gutenberg Project

The Project Gutenberg is a pioneering and expansive digital library initiative focused on encouraging the creation and distribution of eBooks. It was founded by Michael Hart in 1971, making it the oldest digital library. Hart's vision was to make literature more accessible to the public, leveraging the potential of the internet and digital technology to democratize access to books. The project is named after Johannes Gutenberg, the inventor of the movable type printing press, which revolutionized the production of books and made knowledge more widely available in the 15th century, much as Project Gutenberg aims to do in the digital age.

Core Objectives

- **To Provide Free Access to Literature:** Project Gutenberg offers a vast collection of cultural works that are in the public domain, making them freely available to the public.
- **To Preserve Literary Works:** By digitizing and archiving literary works, the project helps preserve literature, including works that might be forgotten or become rare in print form.
- **To Encourage the Creation of eBooks:** The project supports and promotes the creation of eBooks, offering resources for individuals interested in producing digital versions of literary works.

Features and Offerings

- **Extensive Library:** Project Gutenberg's collection includes over 60,000 eBooks as of my last update, comprising a wide range of literary works, including classics, historical texts, and reference materials.
- **Multiple Formats:** eBooks on Project Gutenberg are available in formats that can be read on a variety of devices, including computers, eReaders, smartphones, and tablets. Common formats include plain text, HTML, ePub, Kindle, and PDF.
- **Ease of Access:** Books can be downloaded or read online for free without the need for registration or subscription. This ease of access supports the project's mission of promoting the widespread availability of literature.
- **Volunteer-Driven:** A global network of volunteers carries out the digitization, proofreading, and archiving of books. This community effort ensures the growth and quality of the project's collection.

Impact and Significance

- **Educational Resource:** Project Gutenberg serves as an invaluable resource for students, educators, and researchers, providing free access to a vast repository of literary works.

- **Promotion of Literacy:** By making literature freely available, the project contributes to the promotion of literacy and education worldwide.
- **Preservation of Cultural Heritage:** The digital preservation of literary works ensures that cultural heritage is safeguarded for future generations.

Challenges and Criticisms

While Project Gutenberg has made significant contributions to the availability of digital literature, it also faces challenges such as copyright restrictions that vary by country, impacting the availability of certain works in different regions. Additionally, the quality of digitization can vary, given the reliance on volunteer efforts.

SUMMARY

Project Gutenberg represents a significant effort in the digitization and democratization of literature, embodying the spirit of accessibility and preservation. Its impact extends beyond simply providing free eBooks; it plays a crucial role in educational and cultural preservation efforts, making it a cornerstone of the digital literary landscape.

Why download UTF-8 Plain text files to practice with?

Downloading plain text UTF-8 files into a Linux environment to practice using commands like `find`, `grep`, and `vim` can be an invaluable learning experience for several reasons. This approach provides a practical, hands-on method to enhance your command-line skills, understand text encoding, and manage files efficiently in a Unix-like system. Here's a detailed explanation of why this practice is beneficial:

1. Practicing with Real Content:

Working with real text files, such as those from Project Gutenberg, allows you to apply Linux commands to content that is more varied and complex than simple test data. This can help in understanding how commands work in real-world scenarios, dealing with actual challenges such as varying text formats, large file sizes, or complex data structures within the text.

2. Understanding Text Encoding:

UTF-8 is a widely used encoding format that supports a vast range of characters from different languages, making it universally applicable. By downloading and working with UTF-8 encoded files, you gain practical experience in handling text encoding, which is crucial for processing and presenting text correctly, especially in a multicultural, multilingual digital environment.

3. Mastering `grep` for Text Search:

`grep` is a powerful tool for searching text using patterns. By practicing with `grep` on real text files, you can learn how to effectively use regular expressions to find specific information, such as particular words, phrases, or patterns. This skill is highly valuable for tasks ranging from simple text search to complex data analysis and processing.

4. Using `find` to Locate Files:

The `find` command is essential for navigating large file systems and locating files based on criteria like name, size, or modification date. Practicing with `find` on a collection of downloaded text files teaches you how to efficiently search and manage files in a Linux environment, a crucial skill for system administration and file organization tasks.

5. Editing with `vim`:

`vim` is a powerful text editor that operates in multiple modes and offers a wide array of features, from simple text editing to complex programming tasks. By using `vim` to edit downloaded text files, you not only get to practice text editing but also learn how to use `vim`'s advanced features, such as syntax highlighting, search and replace, and custom configurations. This experience can significantly enhance your text editing efficiency and productivity.

6. Combining Commands for Advanced Tasks:

Linux commands can be combined using pipes (`|`) and redirections (`>`, `>>`) to perform complex tasks on text files. Working with real text content allows you to practice combining commands like `grep`, `sort`, `uniq`, `awk`, and `sed` to filter, sort, and transform text data in powerful ways. This practice can be particularly useful for scripting, data processing, and automation.

7. Hands-on Experience with Linux File System:

Managing files—such as moving (`mv`), copying (`cp`), deleting (`rm`), and changing permissions (`chmod`, `chown`)—is a fundamental aspect of working within a Linux environment. Practicing these file operations on a collection of text files helps solidify your understanding of the Linux file system and its permissions model.

Summary:

Downloading plain text UTF-8 files for practice with commands like `find`, `grep`, and `vim` provides a practical and effective way to deepen your understanding of Linux command-line tools and their application. This approach not only builds your technical skills but also enhances your ability to work with text data in a variety of professional contexts.

Exercise 4: Download and Open a file from the Gutenberg Project

Objective: Download, and then open an existing file named `pg25973.txt` in the current directory using Vim.

Instructions:

1. `wget https://www.gutenberg.org/cache/epub/25973/pg25973.txt`
2. Validate that the file `pg25973.txt` is in your working directory.
3. Use the Vim command to open the file `pg25973.txt`.
4. Once open, navigate through the file using the arrow keys.
5. Exit Vim without making changes by pressing `:q!` and then Enter.

*** If you are struggling, go back to the earlier exercises ***

Exercise 5: Edit a file from the Gutenberg Project

Objective: Open and edit an existing file named `pg25973.txt` in the current directory using Vim.

Instructions:

1. Use the Vim command to open the file `pg25973.txt`.
2. Once open, navigate through the file using the arrow keys.
3. Navigate to where it states, “Title: Birds of the Rockies”
4. Switch from `NORMAL MODE` to `INSERT MODE`
5. Modify, “Title: Birds of the Rockies” to “Title: Birds of the Coasts”
6. Exit vim without making changes by pressing `:q!` and then Enter.

7. Use the Vim command to open the file `pg25973.txt`.
8. Once open, navigate through the file using the arrow keys.
9. Navigate to where it states, “Title: Birds of the Rockies”, validate that if you exit without making changes, it indeed abandons your changes.

10. Switch from `NORMAL MODE` to `INSERT MODE`
11. Modify, “Title: Birds of the Rockies” to “Title: Birds of the Coasts” and exit `INSERT MODE`
12. While in `NORMAL MODE`, save your changes via vim by switching to `COMMAND MODE`
13. While in `COMMAND MODE`, save by typing :w
14. Re-enter `COMMAND MODE`, and exit vim by typing :q

15. Validate your changes by using the head -n 15 pg25973.txt command.

CIRCLING BACK – Saving in VIM

In Vim, one of the most powerful and versatile text editors available in the Linux environment, Command Mode commands allow users to perform a variety of file management and control tasks. The commands `:q!`, `:q`, `:w`, and `:wq` are fundamental for managing Vim sessions, dealing with the saving, exiting, or both actions on the files being edited.

`:q!` (Quit without Saving)

- **Usage:** This command is used to exit Vim without saving any changes made to the document during the current session.
- **Context:** It's particularly useful when you've made changes that you decide not to keep. Executing `:q!` ensures that none of the modifications are saved, allowing you to exit the editor and leaving the file in its original state before the changes.
- **Mechanics:** The exclamation mark (`!`) indicates a forced action, overriding any warnings about unsaved changes.

`:q` (Quit)

- **Usage:** This command is used to quit Vim.
- **Context:** If there have been no changes made to the document since the last save, or if the file was opened but not modified, `:q` will close Vim and return you to the terminal.
- **Constraints:** If unsaved changes exist, Vim will prevent the quit action to protect against potential data loss. In such cases, you must either save the changes before quitting (`:w` then `:q`), quit without saving changes (`:q!`), or use `:wq` to save changes and quit in one command.

`:w` (Write)

- **Usage:** This command saves the current document. It writes the modifications made to the file back to the disk.
- **Context:** Useful for periodically saving your progress without wanting to exit the editor. It ensures that your changes are saved while allowing you to continue working.
- **Options:** Can be combined with a file name (`:w newfile.txt`) to save the current document as a new file, effectively creating a copy without altering the original file unless the file name specified is the same.

`:wq` (Write and Quit)

- **Usage:** This command saves any changes made to the document and then exits Vim.
- **Context:** It's a convenient way to ensure that your work is saved without having to issue separate commands for saving and quitting. It combines the actions of `:w` and `:q` into a single command, streamlining the process of securing your changes and exiting the editor.

- **Mechanics:** Like `:w`, `:wq` can also be followed by a file name to save changes to a new file and exit Vim. If no file name is provided, it saves the changes to the current document and exits.

Summary

Understanding and using these commands efficiently can significantly enhance your productivity in Vim. They offer essential control over your editing session, allowing you to manage the state of your documents effectively. Whether you're making quick edits, working on extensive coding sessions, or just reading through documents, knowing how to properly save, exit, or do both is crucial for a seamless Vim experience.

Exercise 6: Download and Open a file from the Gutenberg Project v2

Objective: Download, and then open an existing file named `pg100.txt` in the current directory using Vim. This is the "Complete Works of William Shakespeare"

https://www.gutenberg.org/cache/epub/100/pg100.txt

1. Open it in vim, edit it, and abandon your changes.
    1. Validate your changes didn't get saved.
2. Open it in vim, edit it, and save your changes.
    1. Validate your changes did get saved.

*** If you are struggling, go back to the earlier exercises ***

SCP – Secure Copy

The `scp` (Secure Copy Protocol) command in Linux is used for securely transferring files and directories between two locations over a network. It leverages the `ssh` (Secure Shell) protocol for data transfer, providing the same level of security and requiring the same authentication that SSH

does. This makes `scp` a preferred choice for transferring files between servers or between a local machine and a server in a secure manner.

`scp` (Secure Copy Protocol) is a command-line tool used for securely transferring files between a local host and a remote host or between two remote hosts. It is built on the `ssh` (Secure Shell) protocol, which provides a secure channel over an unsecured network in a client-server architecture. Understanding how `scp` leverages `ssh`, including its use of standard ports, protocols, and encryption, is key to appreciating its security features.

SSH Protocol Underpinning

`ssh` is a cryptographic network protocol designed for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers. It provides several key features such as authentication, encryption, and data integrity to secure network communications.

When `scp` initiates a file transfer, it effectively sets up an `ssh` session between the source and destination hosts, inheriting SSH's security mechanisms:

1. **Authentication:** `ssh` supports multiple authentication methods, including password authentication and public key authentication. `scp` uses these mechanisms to authenticate the user on the remote host before initiating the transfer. This ensures that only authorized users can transfer files.
2. **Encryption:** `ssh` encrypts the entire session, including the file data being transferred, command execution, and any data exchanged during the session. This means that `scp` transfers are protected from eavesdropping and man-in-the-middle attacks. The encryption algorithms used by `ssh`, such as AES (Advanced Encryption Standard), are industry-standard and widely regarded as secure.
3. **Data Integrity:** `ssh` uses message authentication codes (MACs) to ensure the integrity of the data transmitted. This mechanism detects any alterations to the data during transit, ensuring that the files transferred by scp arrive intact and unmodified.

Standard Ports and Protocols

- **Port 22:** `ssh`, and thus `scp`, typically uses TCP port 22 for communication. This is the default port assigned for `ssh` traffic and is used unless otherwise specified by the user (using the -P option with `scp` to specify a different port).
- **SSH Protocols:** `ssh` operates primarily on the application layer of the network stack, using the underlying TCP protocol for transport. The `ssh` protocol itself consists of three major components:

1. **The Transport Layer Protocol:** Provides server authentication, confidentiality, and integrity with perfect forward secrecy.
2. **The User Authentication Protocol:** Authenticates the client to the server.
3. **The Connection Protocol:** Multiplexes the encrypted tunnel into several logical channels, allowing for file transfers, terminal sessions, and port forwarding over the same `ssh` connection.

Copying multiple text files via `scp` (Secure Copy Protocol)

This can be achieved in a couple of ways, depending on your specific requirements and the structure of your files. `scp` does not natively support specifying multiple distinct source files in a single command to different destinations or even to the same destination directory. However, there are workarounds to achieve the desired outcome.

Method: Using Wildcards for Multiple Files in the Same Directory

If the files you wish to copy are all located in the same directory and can be matched by a pattern, you can use wildcards (`*`) with `scp`.

Example:

To copy all `.txt` files from a local directory to a remote directory, you can use:

```
scp /path/to/local/directory/*.txt user@remotehost:/path/to/remote/directory/
```

This command will match and copy all files with a `.txt` extension from the specified local directory to the specified directory on `remotehost`.

Conclusion

`scp` 's reliance on SSH for file transfers brings several advantages in terms of security. By utilizing SSH's authentication, encryption, and data integrity features, `scp` ensures that file transfers are secure, private, and reliable. The use of standard ports and protocols further integrates `scp` into existing network infrastructures, making it a versatile tool for secure file transfers in a variety of environments.

Basic Syntax – Full Breakdown

The standard syntax of the `scp` command is as follows:

```
scp [OPTION] [user@]SRC_HOST:]file1 [user@]DEST_HOST:]file2
```

1. scp is the command
2. [option] is *IF* you were to use any options they would go here
3. user would be the username you are logging in with
4. @ this is the delimiter between the user name and the source ip
5. : comes right after the source, if you leave it alone it uses the home folder
    a. Or you can use the absolute or relative paths to the file
6. user would be the username you are logging in with
7. @ this is the delimiter between the user name and the destination ip
8. : comes right after the source, if you leave it alone it uses the home folder
    a. Or you can use the absolute or relative paths to the file

Here it is in text:

scp [OPTION] [user@]SRC_HOST:]file1 [user@]DEST_HOST:]file2

This will allow you to copy from a server with OpenSSH installed to another server with OpenSSH installed.

Most of the time though you will use it to copy from a system you're on to and from a server with OpenSSH installed. (While Linux Servers frequently have OpenSSH installed, Windows workstations, Windows Servers, and Mac Workstations typically do not.)

If you are on a system that does not have OpenSSH Server installed, you'll likely be communicating with one that does. If you can `ssh` to it, you can `scp` to it using the same credentials.

The most likely scenario is you either copying to or from a Linux Server to your Windows Host Operating System. Here are the most likely scenarios with examples:

Copying a File from Local to Remote:

scp /path/to/local/file.txt user@remotehost:/path/to/remote/directory

This command copies `file.txt` from the local system to the specified directory on `remotehost` under the specified user's home directory.

This breaks down to

1. scp
2. local file
    a. Options
        i. no folders shown because you are in the present working directory of the file
        ii. the absolute path and file name on the local system
        iii. the relative path and the file name on the local system
3. username
4. @
5. Ip of the remote host
6. :
    a. Options
        i. If you leave the area after the : blank it just drops it into the home directory of the user
        ii. the absolute path and file name on the remote system
        iii. the relative path and the file name on the remote system

Examples of Copying a File from Local to Remote:

Example 1

scp example.txt ajay@192.168.0.35:

This would copy example.txt from your present working directory in windows, and save it to the home folder of 192.168.0.35 as example.txt with the credentials of ajay.

Example 2

scp c:\users\yenri\example.txt ajay@192.168.0.35:/home/ajay/test/exit2.txt

This would copy example.txt using the absolute path in windows, and save it to the absolute path of /home/ajay/test/exit2.txt (note that I changed the file name) 192.168.0.35 as exit2.txt with the credentials of ajay.

Copying a File from Remote to Local:

scp user@remotehost:/path/to/remote/file.txt /path/to/local/directory

This breaks down to

1. scp
2. username
3. @
4. Ip of the remote host
5. :
    a. Options
        i. The file name if its in the home folder to copy from
        ii. The absolute path and file name on the remote system to copy from
        iii. The relative path and the file name on the remote system to copy from
6. local file to be saved
    a. Options
        i. If you just put the file name, it will save it to the present working directory
        ii. the absolute path and file name on the local system to save the file
        iii. the relative path and the file name on the local system from the present working directory to save the file

Examples of Copying a File from Remote to Local:

Example 1

scp ajay@192.168.0.35:example.txt example.txt

This would copy example.txt from your home folder in Linux with an ip of 192.168.0.35, and save it to the present working directory as example.txt with the credentials of ajay.

Example 2

scp ajay@192.168.0.35:/etc/configuration/firewall.conf c:\users\yenric\Documents\firewall_config/firewall.conf

This would copy the firewall.conf file from your absolute path of /etc/configuration in Linux with a ip of 192.168.0.35, and save it to the absolute path of c:\users\yenric\Documents\firewall_config\ as firewall.conf with the credentials of ajay.

VIDEOS on SCP:

https://www.youtube.com/watch?v=Aa7tKMmeFZI

https://www.youtube.com/watch?v=q2OHvlr081s

Exercise 7: Download and Copy a file from the Gutenberg Project to Linux

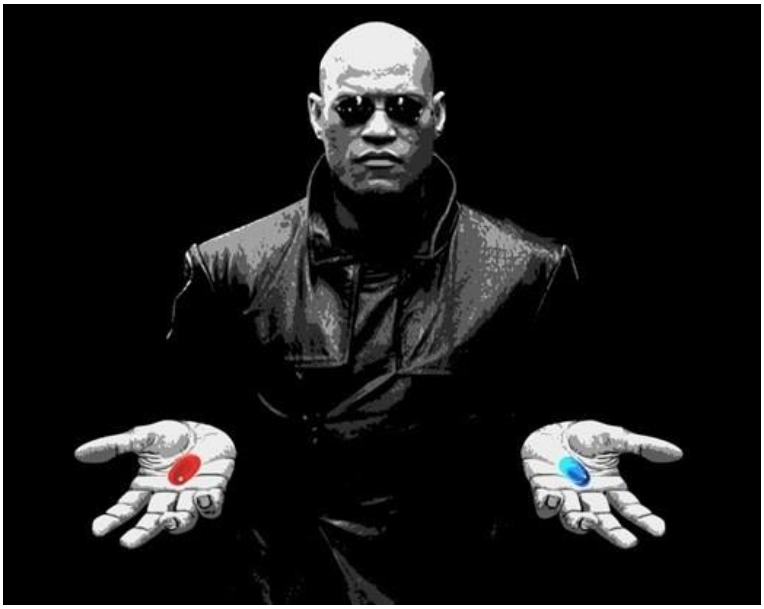
Objective: Download `pg100.txt` in Windows and then copy it out to your Linux server via `scp`.

Exercise 8: Create a text file and then copy it to Windows

Objective: Create a file via `vim` and then copy it out to your Windows workstation via `scp`.

Exercise 9: Create a text file and then copy it to Windows

Objective: Ensure you have 3 or more text files in your Linux home folder, and then copy it to Windows workstation via `scp` with one command.



This is all fundamental

You are expected to know this, have it memorized, and be able to use it

- `scp`
- `ssh`
- `vim`

These commands as well as copy, move, delete, etc are all fundamental in Linux and Windows.

Moving forward you will be expected to be able to use these at the drop of the hat.