H1 Session 2

Session 2 Linux basics

Sure, here are some Linux help notes:

```
1. **Command Basics:**
   - `ls`: List directory contents.
   - `cd`: Change the current directory.
   - `pwd`: Print the current working directory.
   - `mkdir`: Create a directory.
   - `rm`: Remove files or directories.
   - `cp`: Copy files or directories.
   - `mv`: Move or rename files or directories.
   - `nano` or `vim`: Text editors for creating or editing files.

2. **File System Navigation:**
   - Use `cd` followed by the directory name to navigate through the file system.
   - Use `cd ..` to move up one directory.
   - Use `cd ~` to go to the home directory.

3. **File Permissions:**
   - Use `chmod` to change file permissions.
   - `chmod +x filename` to make a file executable.
   - `chmod 755 filename` to give read, write, and execute permissions to the
owner and read and execute permissions to group and others.

4. **Package Management:**
   - On Debian-based systems (like Ubuntu), you can use `apt` or `apt-get` for
package management.
   - On Red Hat-based systems (like CentOS), you can use `yum` or `dnf`.

5. **Process Management:**
   - `ps`: Show information about running processes.
   - `top`: Display and update sorted information about processes.
   - `kill`: Terminate processes.
   - `killall`: Terminate processes by name.

6. **Networking:**
   - `ifconfig` or `ip addr`: Display network interface configuration.
   - `ping`: Test the reachability of a host.
   - `netstat`: Display network connections, routing tables, interface statistics,
masquerade connections, and multicast memberships.
   - `ssh`: Secure shell for remote login.

7. **File Compression:**
   - `tar`: Archive files.
   - `gzip`: Compress or expand files.
   - `zip` and `unzip`: Package and unpack files in zip format.
```

```
8. **User Management:**
   - `useradd`: Add a new user.
   - `passwd`: Change user password.
   - `userdel`: Delete a user.
   - `su`: Switch user.
   - `sudo`: Execute a command as another user, typically the superuser.

9. **System Information:**
   - `uname -a`: Display system information.
   - `df -h`: Show disk space usage.
   - `free -m`: Display amount of free and used memory in the system.

10. **System Updates:**
    - `apt update` or `yum update` to update package lists.
    - `apt upgrade` or `yum upgrade` to upgrade installed packages.
    -
11. **Connect to server**
    1. ssh -i /path/to/private_key_file username@hostname_or_ip
set permissions to private key
    chmod 400 /path/to/private_key_file
```

`ls` lists files and directories in the current directory. ls -a Listing files in directory including hidden `ls -l` lists detailed information about files in the current directory.

Differentiating between files and directories

```
lrwxr-xr-x  1 user user     10 Apr 13 10:00 link_to_file -> target_file
drwxr-xr-x  2 user user   4096 Apr 13 10:00 directory_name
```

```
link_to_file is a symbolic link (l at the beginning).
directory_name is a directory (d at the beginning).
```

you can differentiate between files and directories by using the ls command with the -F option. This option adds a trailing indicator to each listed item, denoting its type:

```
/: Denotes directories.
*: Denotes executable files.
@: Denotes symbolic links.
|: Denotes FIFOs (named pipes).
=: Denotes sockets.
```

`cd / cd ~` navigate to the home directory in one command using cd, you can use the cd command without any arguments or with the ~

`ls -la` lists all files (including hidden ones) in a detailed format (`-l`), and it does so in alphabetical order (`-a`).

`ls -lr` lists files in reverse order (`-r`) of their names without any detailed information. `ls -lt` lists files in the current directory sorted by modification time, with the newest files appearing at the top.

`ls -ltr` lists files in the current directory sorted by modification time in reverse order, with the oldest files appearing at the top. `ls -la` lists all files (including hidden ones) in a detailed format.

File creation

touch: This command is used to create empty files or update the access and modification timestamps of existing files. Syntax:

touch filename

echo: This command is used to display a line of text/string that is passed as an argument. It can also be used to create a new file and add content to it. Syntax to create a new file:

echo "content" > filename

cat: This command is primarily used to display content of files. However, it can also be used to create new files and add content to existing ones. Syntax to create a new file:

cat > filename

when you run the command echo "new content" > filename, it will update filename with the new content specified in the echo command. If filename doesn't exist, a new file with that name will be created.

appending and overwriting file `>` and `>>` are both redirection operators used in conjunction with the `cat` command to manipulate file content:

- `>`: Redirects the output of a command to a file, overwriting the existing content of the file. If the file does not exist, it creates a new file. Example:

  ```
  cat file1 > file2
  ```

  This command reads the content of `file1` and writes it to `file2`, overwriting any existing content in `file2`.

- `>>`: Redirects the output of a command to a file, appending it to the end of the file. If the file does not exist, it creates a new file. Example:

  ```
  cat file1 >> file2
  ```

  This command reads the content of `file1` and appends it to `file2`, preserving the existing content in `file2`.

In summary, `>` overwrites the content of the target file, while `>>` appends to it.

removing file and directories To remove a file or directory in Linux, you can use the `rm` command for files and the `rmdir` command for directories. Here's the one-liner examples for each:

Remove a file:

```
rm filename
```

Remove a directory (which must be empty):

```
rmdir directory_name
```

If the directory is not empty and you want to remove it along with its contents, you can use the `-r` or `-rf` option with the `rm` command. However, be cautious when using the `-rf` option, as it forcefully removes files and directories without asking for confirmation and can lead to data loss if used incorrectly.

file management -copy move

To copy files and directories in Linux, you can use the `cp` command.

Here's the one-liner examples for both:

To copy a file:

```
cp source_file destination_file
```

To copy a directory and its contents recursively:

```
cp -r source_directory destination_directory
```

Replace `source_file` and `destination_file` with the appropriate file names, and `source_directory` and `destination_directory` with the appropriate directory names.

The `-r` option is used to copy directories recursively, including all subdirectories and their contents. Without this option, `cp` will only copy individual files and not directories.

To move files or directories in Linux, you can use the `mv` command.

Here's the one-liner examples for both:

To move a file:

```
mv source_file destination_directory
```

To move a directory:

```
mv source_directory destination_directory
```

Replace `source_file` with the file you want to move and `destination_directory` with the directory where you want to move the file. Similarly, replace `source_directory` with the directory you want to move and `destination_directory` with the destination directory.

If you want to rename the file or directory while moving, you can specify the new name in the destination:

```
mv old_name new_name
```

This command will rename `old_name` to `new_name`. If `new_name` is a directory, `old_name` will be moved into it.

what is passwd file

The `/etc/passwd` file is a text-based database in Unix-like operating systems (including Linux) that stores essential information about user accounts. Each line in the file represents a single user account and contains seven fields separated by colons (`:`). These fields typically include:

1. **Username**: The name of the user.
2. **Password**: Historically, this field contained an encrypted password, but modern systems store an "x" here and store the encrypted password in the `/etc/shadow` file for security reasons.
3. **User ID (UID)**: A unique numeric identifier for the user.
4. **Group ID (GID)**: The primary group ID associated with the user.
5. **User Info**: Additional information about the user (often the full name or description).
6. **Home Directory**: The path to the user's home directory.
7. **Shell**: The user's default shell (command interpreter).

Here's an example entry in the `/etc/passwd` file:

```
john:x:1001:1001:John Doe:/home/john:/bin/bash
```

- Username: `john`
- Password: `x` (password stored in `/etc/shadow`)
- UID: `1001`
- GID: `1001`
- User Info: `John Doe`
- Home Directory: `/home/john`
- Shell: `/bin/bash`

The `/etc/passwd` file is typically read-only for regular users and accessible only by privileged users (such as the root user) for security reasons. It is used by various system utilities and commands to map user names to user IDs and vice versa.

grepping text you need from à file to get some specific text in a file If you want to search for a specific pattern or text within the output of the `cat` command, you can use the `grep` command.

Here's a one-liner example:

```
cat filename | grep "pattern"
```

Replace `filename` with the name of the file you want to search in, and `pattern` with the text you're searching for.

For example, if you want to search for the word "example" in a file named `file.txt`, you would use:

```
cat file.txt | grep "example"
```

This command will display all lines in `file.txt` that contain the word "example".

appending text to a file

echo "Devops" >> /etc/passwd This command will append the text "Devops" as a new line at the end of the /etc/passwd file.

` cat >> passwd Devops

vs

echo "Devops" >> /etc/passwd

The command `echo "Devops" >> /etc/passwd` appends the text "Devops" to the end of the file `/etc/passwd`. It's a straightforward way to add text to a file.

On the other hand, `cat >> passwd` is not quite the same. It's an attempt to use the `cat` command to append text to a file named `passwd`, but it's missing the actual text to append. This command puts the shell in a mode where it waits for you to input text manually, and it will continue to append whatever you type until you signal the end of input (usually with Ctrl+D). So, if you input "Devops" after running `cat >> passwd` and then press Ctrl+D, it will append "Devops" to the file named `passwd`.

In summary, `echo "Devops" >> /etc/passwd` is the appropriate command to append the text "Devops" to the file `/etc/passwd`, while `cat >> passwd` followed by manual input can achieve a similar result but requires additional steps.

searching for text

To search for the text "Devops" in a file using `grep`, you can use the `-w` option to match whole words and `-i` option for case-insensitive search. Here are the commands:

For case-sensitive search:

```
grep -w "Devops" filename
```

For case-insensitive search:

```
grep -wi "devops" filename
```

Replace `filename` with the name of the file you want to search in.

- `-w` option ensures that only whole words are matched. For example, it won't match "DevopsXYZ".
- `-i` option performs a case-insensitive search, so it will match "Devops", "devops", "DEVOPS", etc.

If you want to match partial text as well, you can remove the `-w` option. Here's the command for both case-sensitive and case-insensitive searches:

For partial match with case-sensitive search:

```
grep "Devops" filename
```

For partial match with case-insensitive search:

```
grep -i "devops" filename
```

These commands will search for the text "Devops" (or "devops" in case-insensitive mode) anywhere in the file, including partial matches.

Piping in Linux allows you to redirect the output of one command as the input of another command, creating a chain of commands to perform complex tasks.

The pipe symbol `|` is used to connect the output of one command to the input of another command. Here's the basic syntax:

```
command1 | command2
```

This means that the output of `command1` will be passed as input to `command2`.

For example, to list all files in the current directory and then search for a specific pattern within those files, you can use:

```
ls | grep "pattern"
```

This command will list all files in the current directory using `ls`, and then the output of `ls` will be passed as input to `grep` to search for the specified pattern.

Piping is a powerful feature in Linux that allows you to combine simple commands to perform complex tasks efficiently.

wget vs curl `wget` and `curl` are both command-line tools used to transfer data over various network protocols. Here's a brief comparison:

- **wget:** It is a command-line utility for downloading files from the web. It supports HTTP, HTTPS, and FTP protocols. `wget` is simple to use and is primarily focused on downloading files. It provides options for recursive downloads, resuming downloads, and downloading multiple files.

- **curl:** It is a more versatile command-line tool that supports a wide range of protocols, including HTTP, HTTPS, FTP, FTPS, SCP, SFTP, LDAP, TELNET, SMTP, POP3, IMAP, and many more. In addition to downloading files like `wget`, `curl` can also upload files, send requests to web servers, and perform various other tasks. It is often used for testing APIs and performing HTTP requests with specific headers and data payloads.

In summary, if you need a simple tool for downloading files from the web, `wget` is a good choice. However, if you require more advanced features or need to work with a wider range of protocols, `curl` is the better option.

cut vs awk `cut` and `awk` are both command-line tools used for text processing in Unix-like operating systems. Here's a brief comparison:

- **cut:** `cut` is primarily used for extracting specific columns or fields from text files. It is particularly useful when dealing with files that have fixed-width columns or fields separated by a delimiter (such as a space or a comma). `cut` is simple and efficient for basic text extraction tasks.

- **awk:** `awk` is a more powerful and versatile tool for text processing. It is a complete programming language in itself, allowing you to perform complex operations on text files. With `awk`, you can perform tasks such as searching for patterns, filtering data based on conditions, performing calculations, and formatting output. `awk` is particularly useful for tasks that require more advanced text processing and data manipulation.

In summary, if you need to quickly extract specific columns or fields from a text file, `cut` is a good choice. However, if you require more advanced text processing capabilities or need to perform complex operations on text data, `awk` is the better option.

Sure, here are examples of both `cut` and `awk` with explanations:

cut:

Suppose you have a file called `data.txt` with the following content:

```
Name     Age    Gender
John     25     Male
Jane     30     Female
Alice    22     Female
```

If you want to extract only the "Name" column from this file, you can use `cut` as follows:

```
cut -d' ' -f1 data.txt
```

Explanation:

- `-d' '` specifies the delimiter used in the file. In this case, it's a space.
- `-f1` specifies the field (column) number to extract, which is the first column.

This command will output:

```
Name
John
Jane
Alice
```

awk:

Using the same `data.txt` file, let's say you want to extract only the rows where the age is greater than 25. You can use `awk` as follows:

```
awk '$2 > 25' data.txt
```

Explanation:

- `$2` refers to the second field (column) in each row, which is the "Age" column.
- `> 25` specifies the condition that the age must be greater than 25.

This command will output:

```
Jane      30     Female
```

In summary, `cut` is useful for simple text extraction tasks where you know the position or delimiter of the fields you want to extract, while `awk` is more powerful and flexible, allowing you to perform complex operations and conditions on text data.

example of awk

awk -F / '{print $1F}' awk -F / '{print $NF}'

Certainly! Here's an explanation of the two `awk` commands:

1. `awk -F / '{print $1F}'`:
   - `-F /`: This option sets the field separator to `/`. It tells `awk` to split each input line into fields based on the `/` character.
   - `'{print $1F}'`: This statement instructs `awk` to print the first field (`$1`) followed by the letter "F".

So, if the input is `https://github.com/pramod-k-codes/devops-daws76s-notes/blob/master/session-02.md`, this command will print `https://github.com/pramod-k-codes/devops-daws76s-notes/blob/master/session-02.mdF`.

2. `awk -F / '{print $NF}'`:

   - `-F /`: This option sets the field separator to `/`, similar to the first command.
   - `'{print $NF}'`: This statement instructs `awk` to print the last field (`$NF`), where `NF` is a special variable in `awk` representing the number of fields.

   So, if the input is `https://github.com/pramod-k-codes/devops-daws76s-notes/blob/master/session-02.md`, this command will print `session-02.md`.

In summary, both commands use `awk` to extract specific fields from input lines separated by `/`, with the first command printing the first field followed by the letter "F" and the second command printing the last field.

head and tail `head filename` displays the first 10 lines of the specified file, while `tail filename` displays the last 10 lines. If you want to display a different number of lines, you can use the `-n` option followed by the number of lines you want to display.

For example:

- `head -n 20 filename` will display the first 20 lines of the file.

- `tail -n 5 filename` will display the last 5 lines of the file.

  head -n 3 filename will display the first 3 lines of the file named filename. command | head -n 3 will display the first 3 lines of the output of the command. **echo vs cat**

`echo` and `cat` are both commands used to display text in the terminal, but they serve different purposes:

- **echo:** This command is used to print a line of text to the standard output. It is commonly used to display messages, variables, or command output in scripts or the terminal.

- **cat:** This command is used to concatenate and display the contents of one or more files. It is often used to display the content of text files or to combine multiple files into a single output.

In summary, `echo` is used to display specified text, while `cat` is used to display the contents of files.

vim options

After typing `:` in Vim, you enter the command-line mode, where you can execute various commands. Here are some common options you have after typing `:`:

1. **Save and Quit Commands:**

   - `w`: Save the current file.
   - `q`: Quit Vim.
   - `wq` or `x`: Save and quit.
   - `q!`: Quit without saving (force quit).
   - `wq!` or `x!`: Save and quit (force save and quit).

2. **Editing Commands:**

   - `e {file}`: Open a new file for editing.
   - `bn` or `:bnext`: Switch to the next buffer.
   - `bp` or `:bprevious`: Switch to the previous buffer.
   - `bd` or `:bdelete`: Close the current buffer.

3. **Search and Replace Commands:**

   - `/{pattern}`: Search for a pattern.
   - `:s/{pattern}/{replacement}/`: Replace the first occurrence in the current line.
   - `:s/{pattern}/{replacement}/g`: Replace all occurrences in the current line.
   - `:s/{pattern}/{replacement}/gc`: Replace all occurrences in the current line with confirmation.

4. **Navigation Commands:**

   - `:set number`: Toggle line numbers.
   - `:set hlsearch`: Toggle search highlighting.
   - `:set ignorecase`: Ignore case when searching.
   - `:set noignorecase`: Do not ignore case when searching.
   - `:set syntax= {syntax}`: Set syntax highlighting (e.g., `syntax=python`).

5. **Other Commands:**

   - `:help {topic}`: Open help for the specified topic.
   - `:version`: Display version information about Vim.
   - `:qall`: Quit all buffers and exit Vim.

These are just a few examples of the many commands available in Vim's command-line mode. You can explore more commands and options in Vim's documentation (`:help`). Sure, here's an explanation of each option in Vim along with examples:

1. `Shift + g`:

   - Explanation: Moves the cursor to the last line of the file.
   - Example: Pressing `Shift + g` moves the cursor to the last line of the file.

2. `gg`:

   - Explanation: Moves the cursor to the first line of the file.
   - Example: Typing `gg` moves the cursor to the first line of the file.

3. Searching for text "bin":

   - Explanation: Use `/bin` to search for the text "bin" in the file.
   - Example: Typing `/bin` and pressing Enter searches for the text "bin" in the file.

4. Undo last action:

   - Explanation: Press `u` to undo the last action.
   - Example: Typing `u` undoes the last action performed in the file.

5. Undoing highlight:

  - Explanation: Use `:nohlsearch` to disable search highlighting.
  - Example: Typing `:nohlsearch` and pressing Enter disables search highlighting.

6. Finding text and replacing (search "bin" and replace with "bin1") at cursor and in entire file:

   - Explanation: Use `:%s/bin/bin1/g` to replace "bin" with "bin1" in the entire file.
   - Example: Typing `:%s/bin/bin1/g` and pressing Enter replaces "bin" with "bin1" in the entire file.

7. Find and replace in specific line, for example, the 3rd line:

   - Explanation: Use `:3s/bin/bin1/g` to replace "bin" with "bin1" in the 3rd line.
   - Example: Typing `:3s/bin/bin1/g` and pressing Enter replaces "bin" with "bin1" in the 3rd line.

8. Copy and paste line using `yy` and `p`, and paste 10 times using `10p`:

   - Explanation: Use `yy` to copy a line, `p` to paste after the current line, and `10p` to paste 10 times.
   - Example: Typing `yy` copies the current line, `p` pastes the copied line after the current line, and `10p` pastes the copied line 10 times after the current line.

no highlight in vim

`nohl` and `nohlsearch` are both commands used to turn off search highlighting in Vim, but they have slightly different effects:

- `nohl`: This command turns off search highlighting temporarily, but it does not affect the `'hlsearch'` option. If you perform another search, the highlighting will be enabled again.
- `nohlsearch`: This command turns off search highlighting and also sets the `'hlsearch'` option to off, so search highlighting remains disabled until you explicitly turn it back on.

In summary, `nohl` turns off search highlighting temporarily, while `nohlsearch` turns off search highlighting and disables it until re-enabled.

chown changing ownership

Certainly! Here are examples demonstrating the usage of each option with the `chown` command:

1. `-R`: Recursively change ownership of directories and their contents:

   ```
   chown -R username:groupname directory/
   ```

   This command recursively changes the ownership of the directory and all its contents to the specified `username` and `groupname`.

2. `-v`: Verbose mode, display information about files as they are processed:

   ```
   chown -v username:groupname file1 file2
   ```

   This command changes the ownership of `file1` and `file2` to the specified `username` and `groupname`, displaying verbose information about each file as it is processed.

3. `-c`: Display information only about files that are changed:

   ```
   chown -c username:groupname file1 file2
   ```

   This command changes the ownership of `file1` and `file2` to the specified `username` and `groupname`, displaying information only for files that are actually changed.

4. `--reference=<file>`: Set ownership to match that of the specified reference file:

   ```
   chown --reference=reference_file file1
   ```

   This command sets the ownership of `file1` to match that of the `reference_file`.

5. `--from=<oldowner>[:<oldgroup>]`: Change ownership only if the current owner and/or group match the specified values:

   ```
   chown --from=oldowner:oldgroup newowner:newgroup file1
   ```

   This command changes the ownership of `file1` from `oldowner:oldgroup` to `newowner:newgroup`, but only if the current ownership matches `oldowner:oldgroup`.

These examples demonstrate how to use each option with the `chown` command for various scenarios.