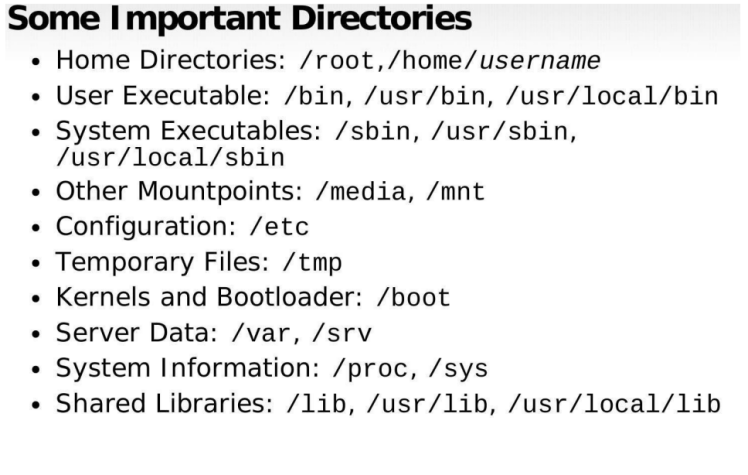
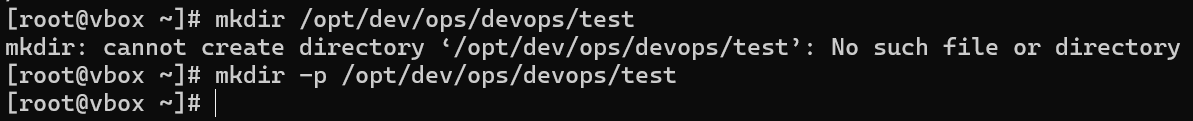
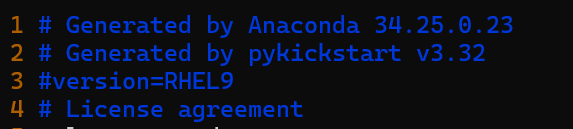
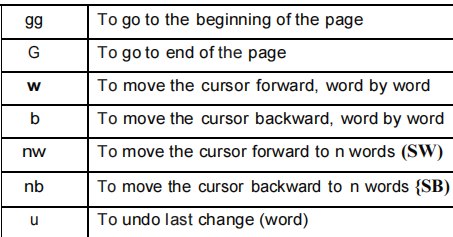
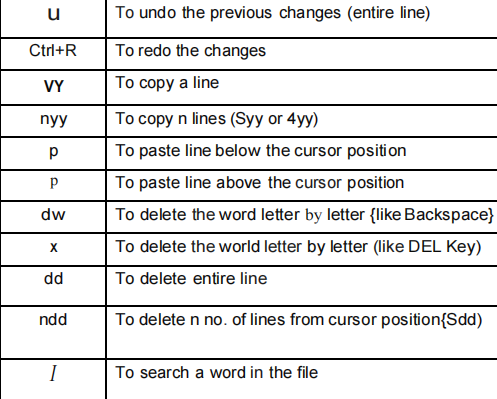
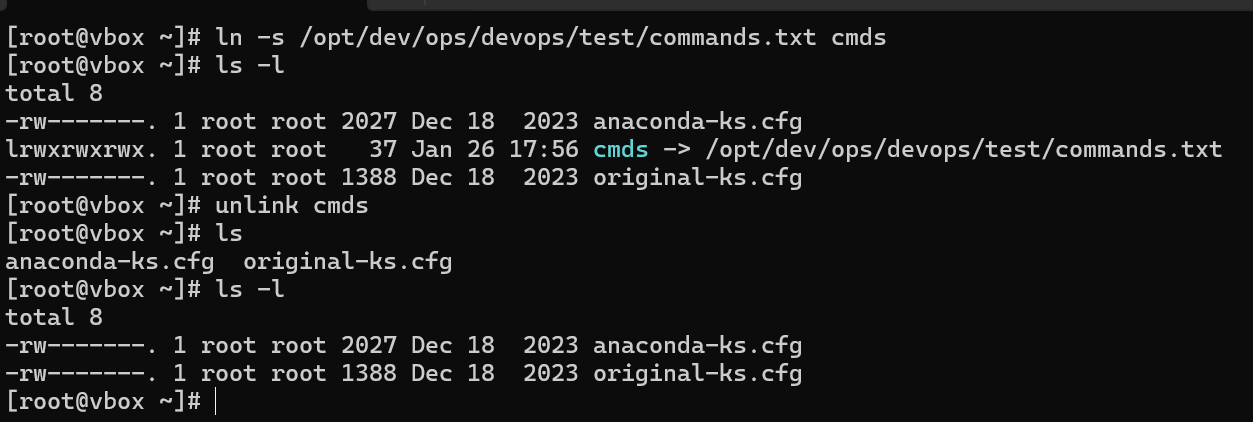
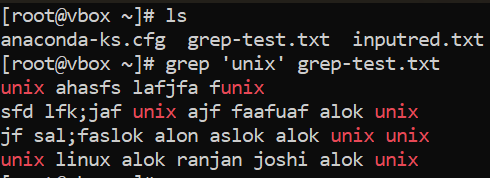
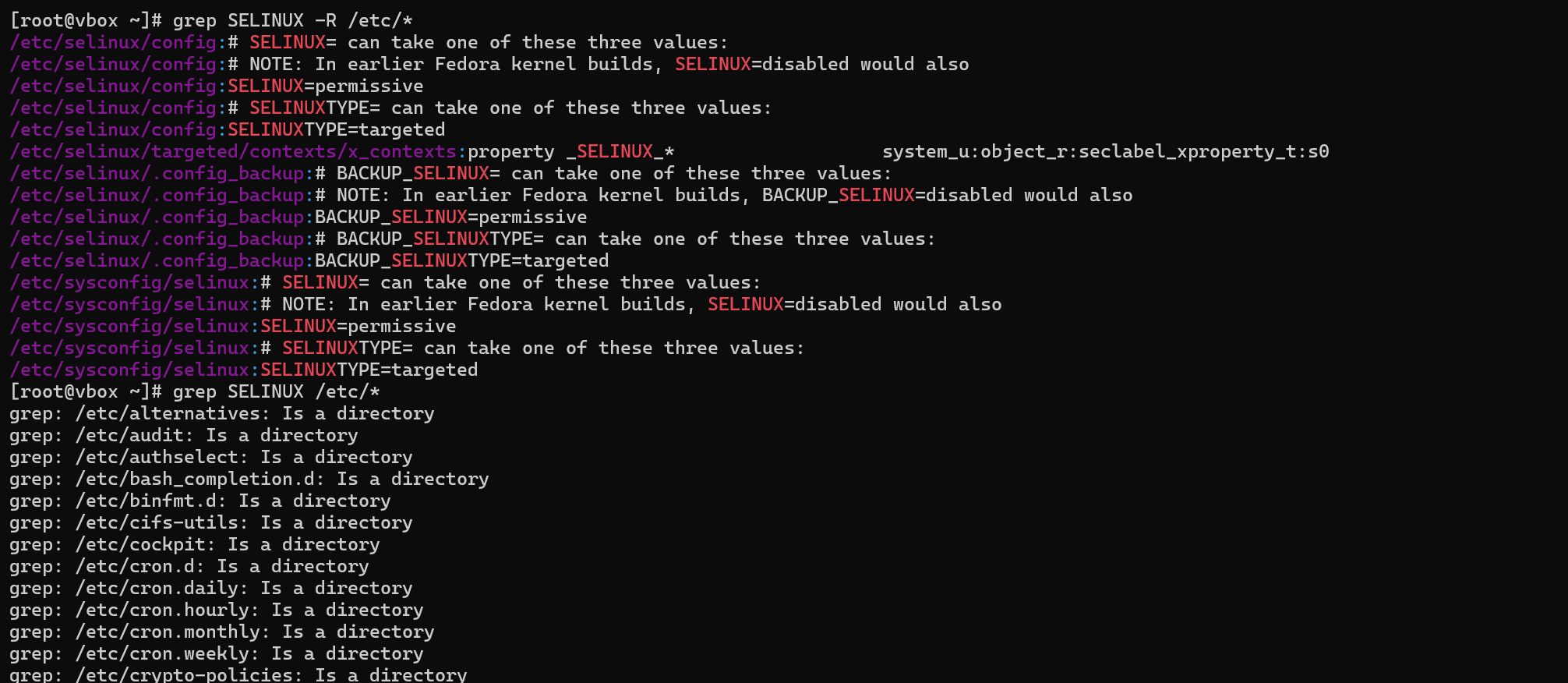
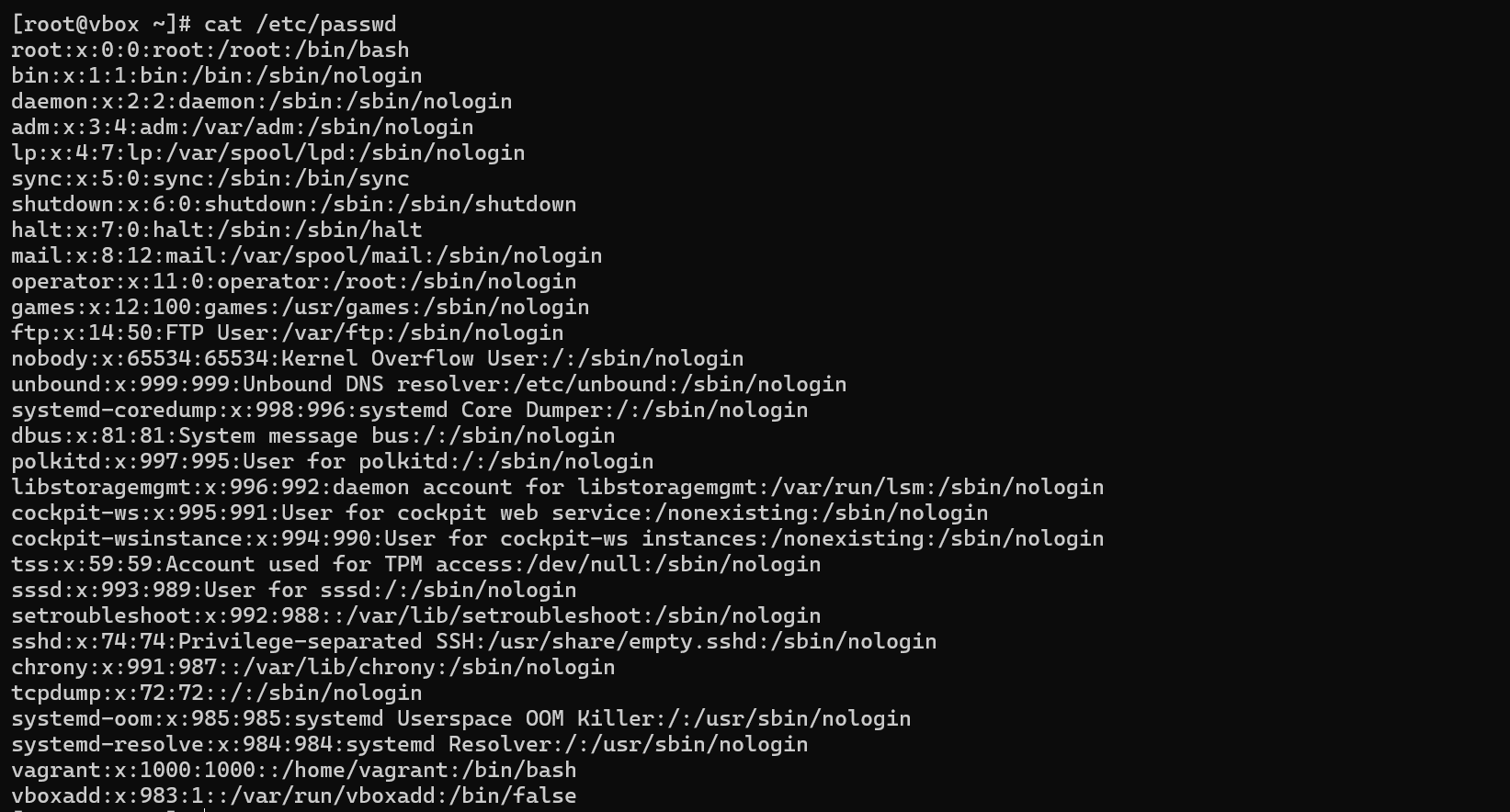
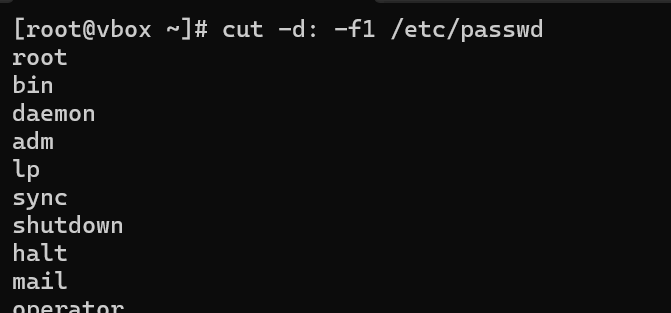
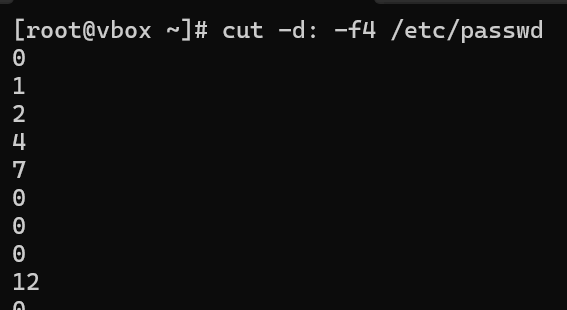
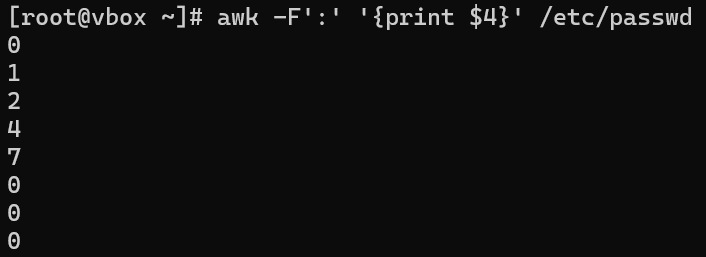
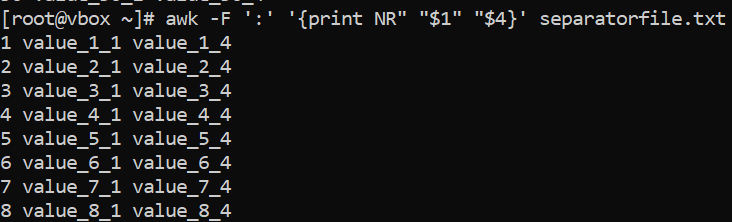
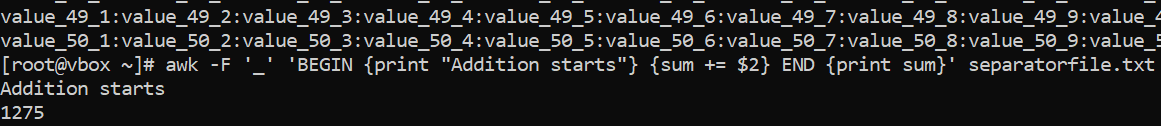
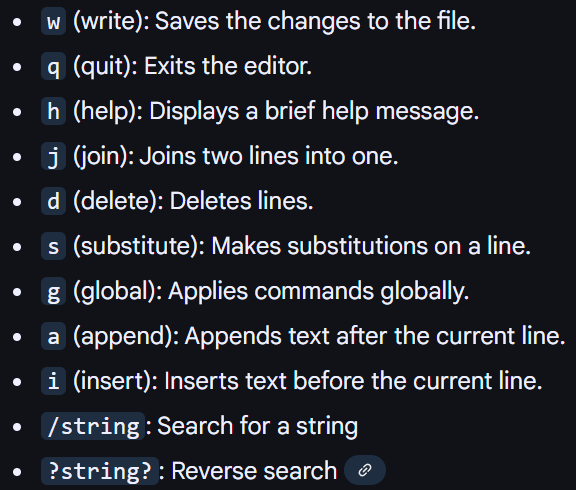
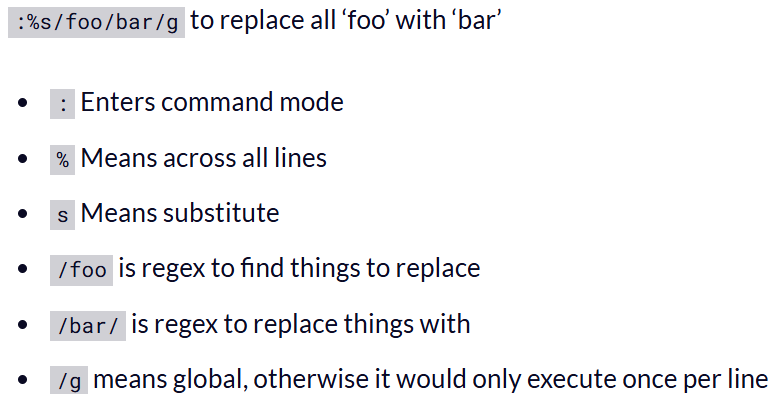
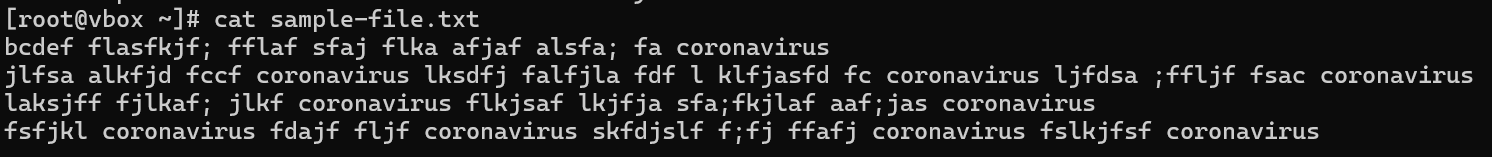
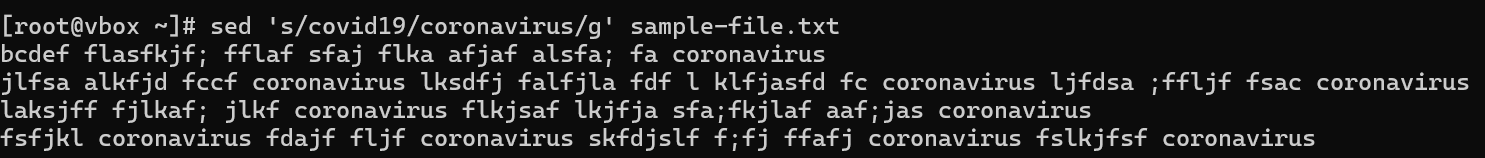
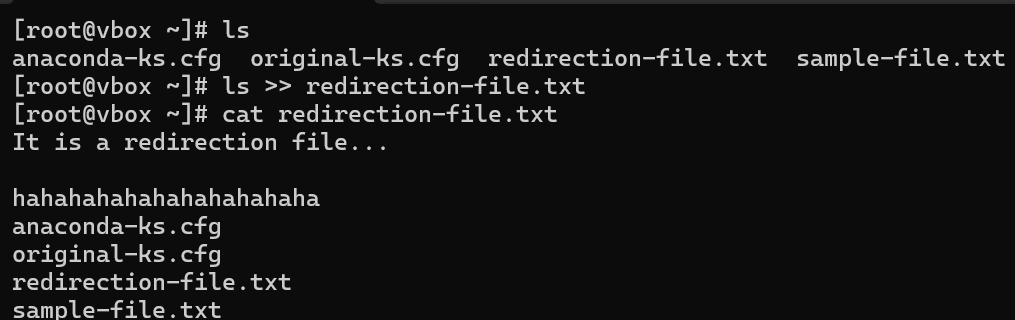
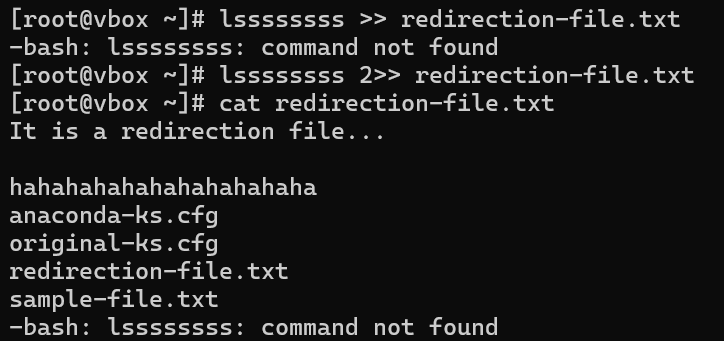
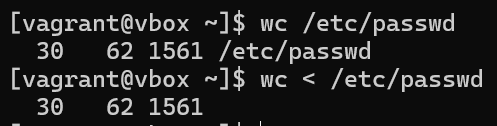
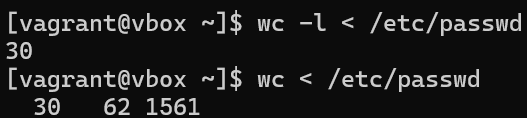
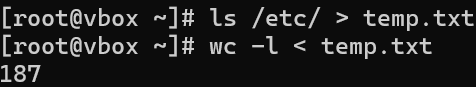
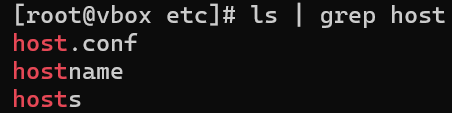
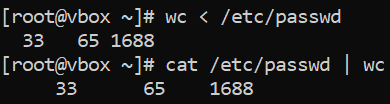
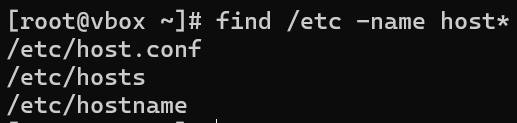
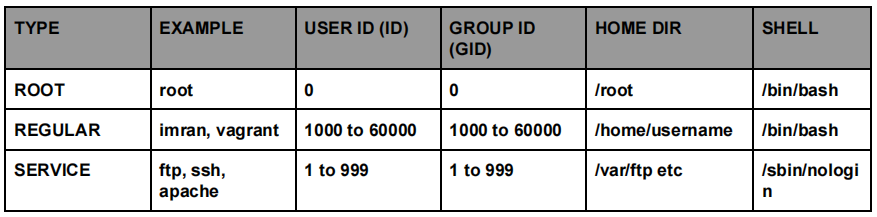
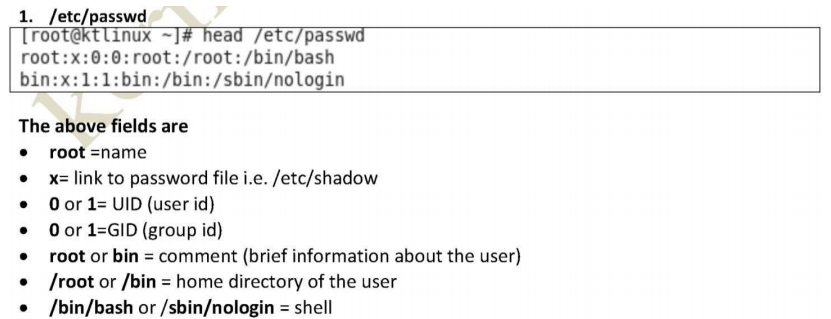
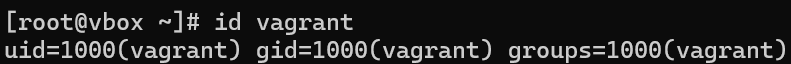
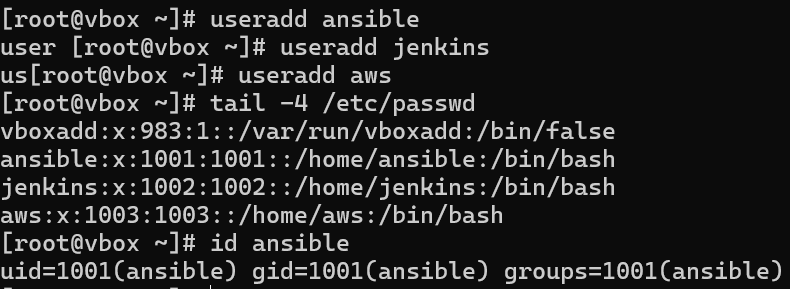
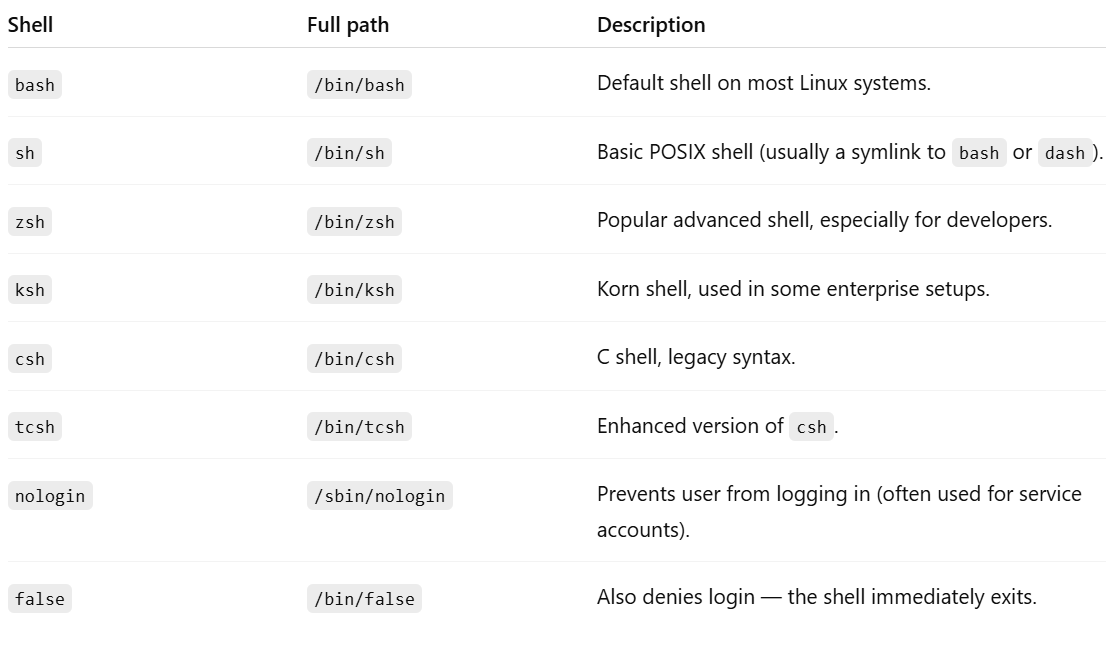
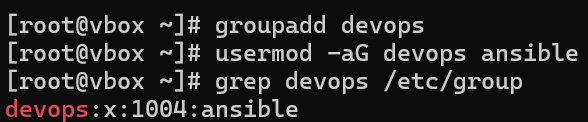
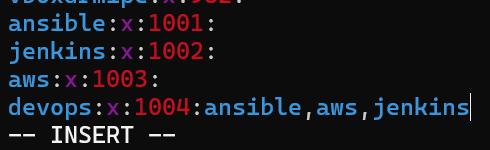
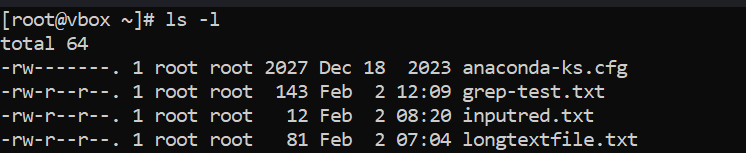
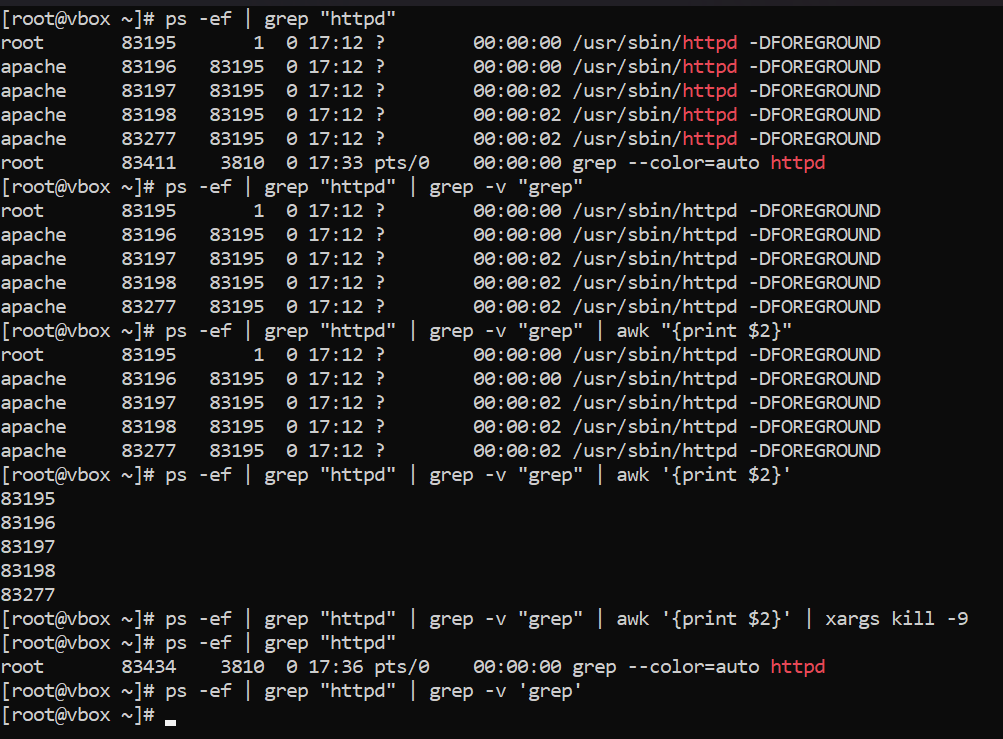
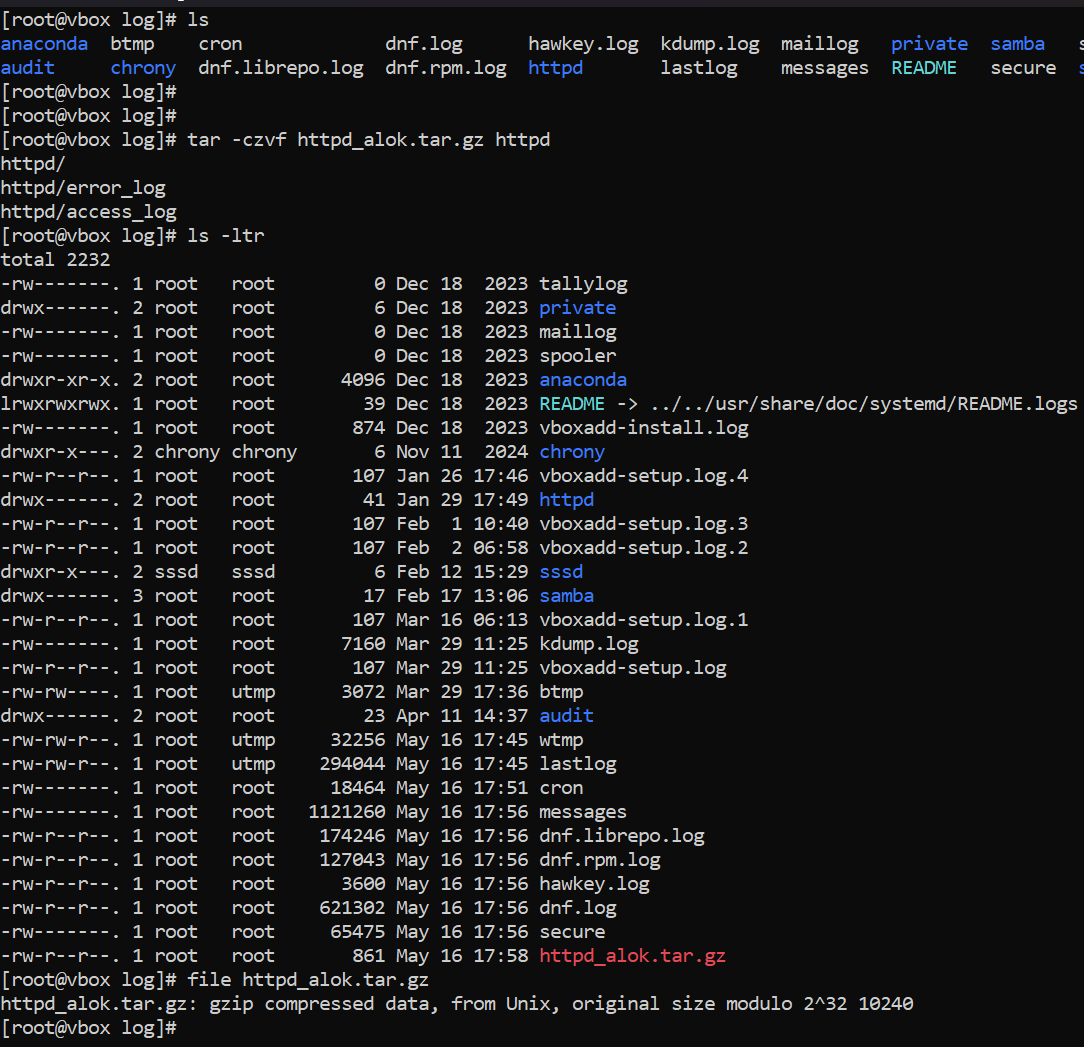
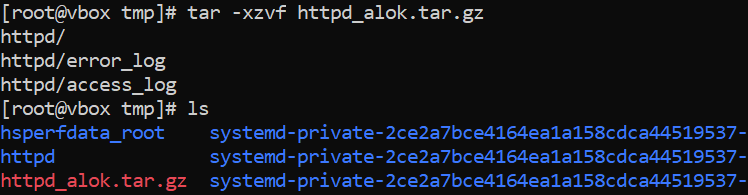
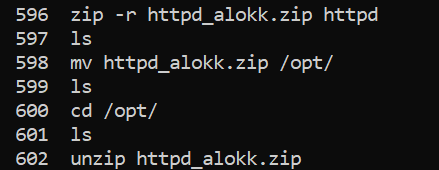
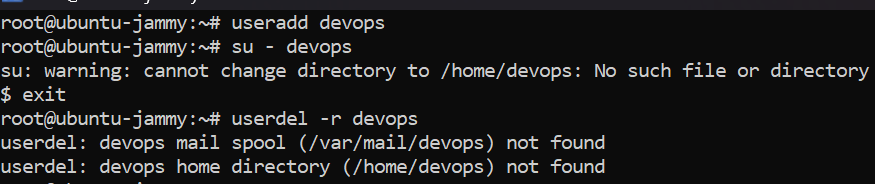
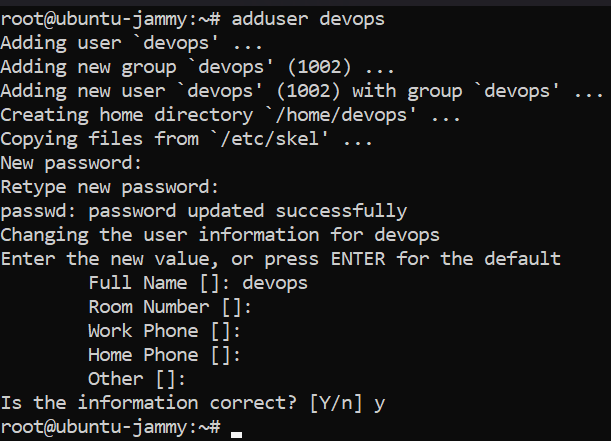
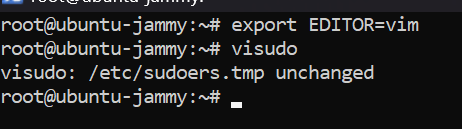
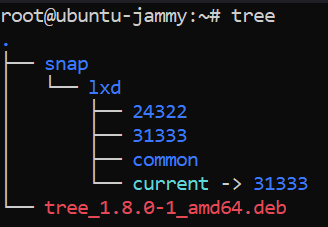
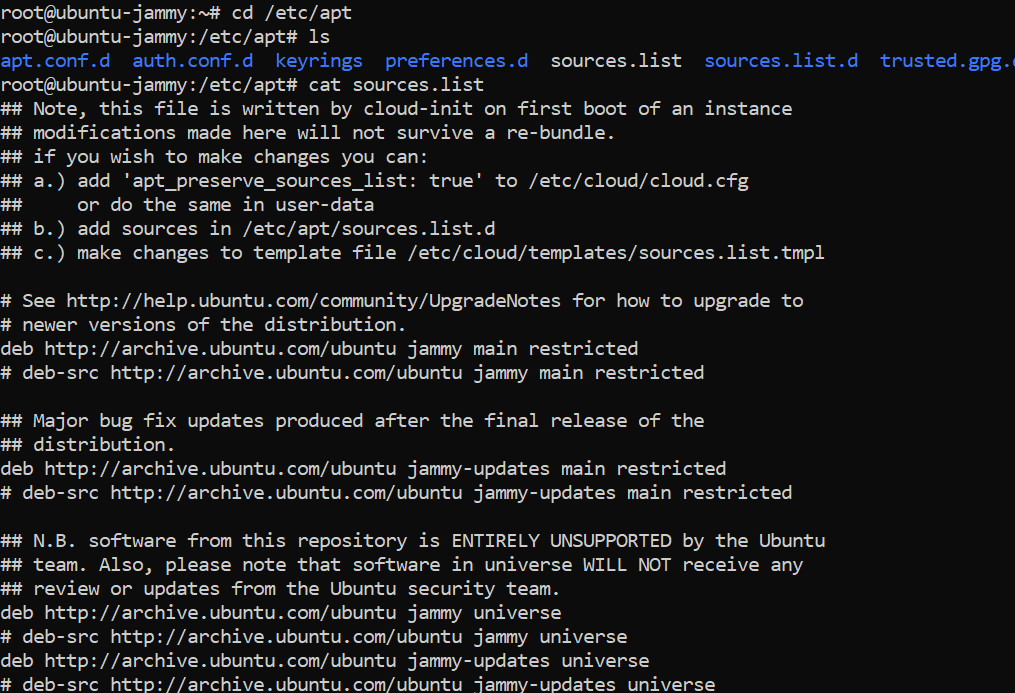
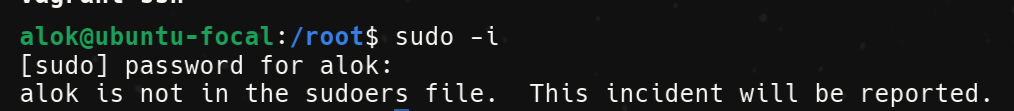
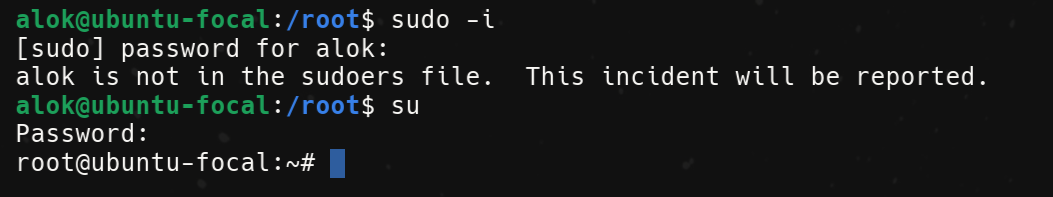
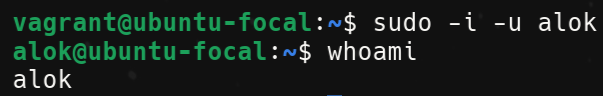
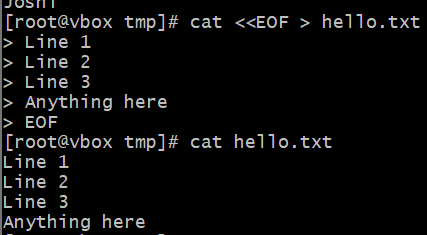
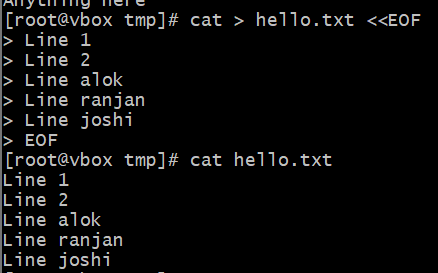
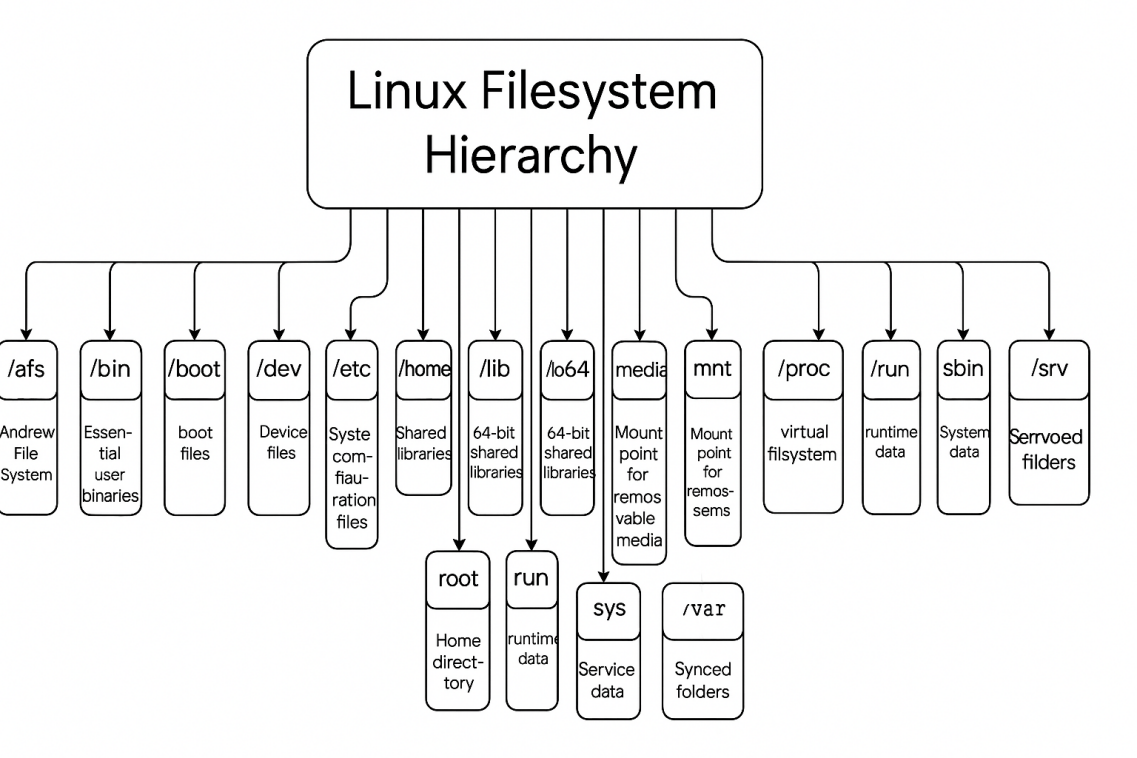
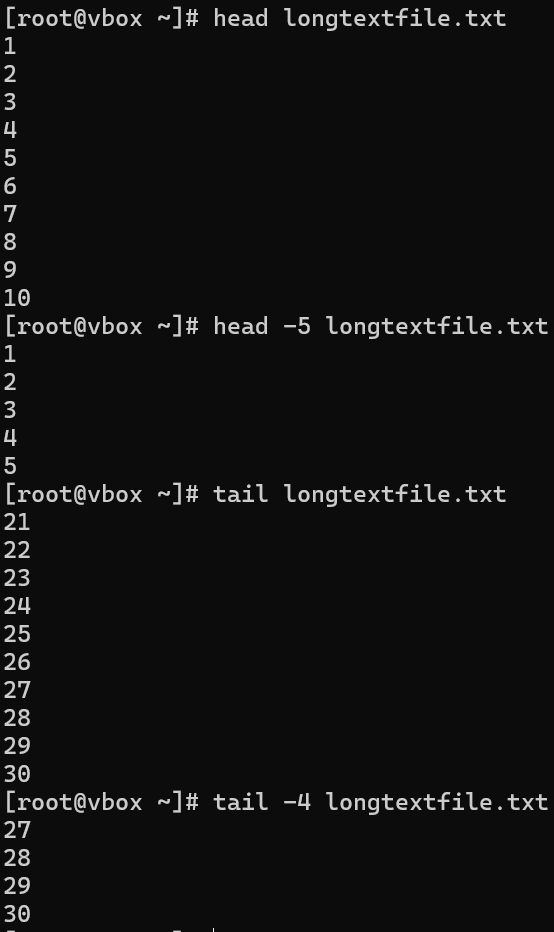
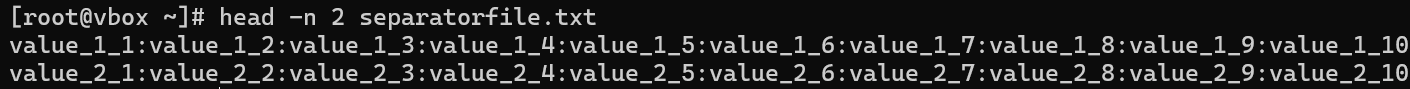
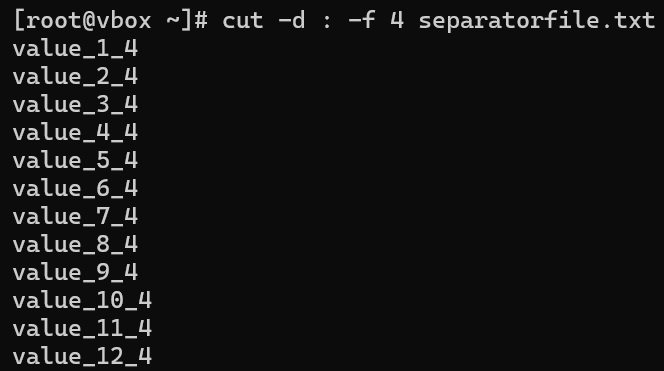
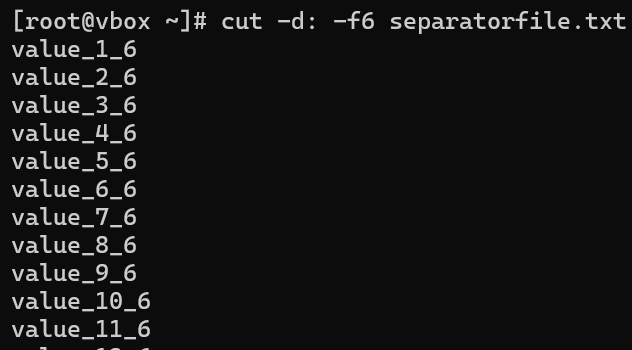
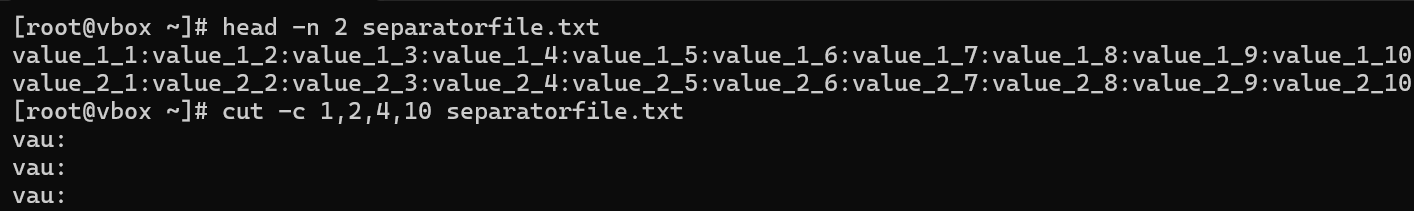
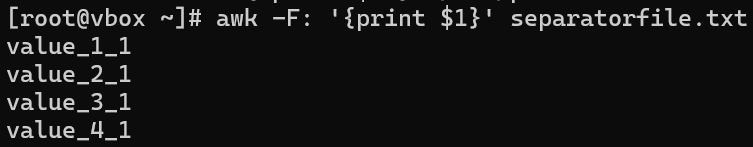
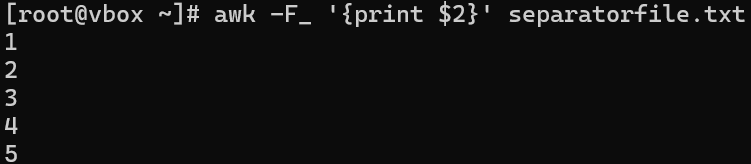
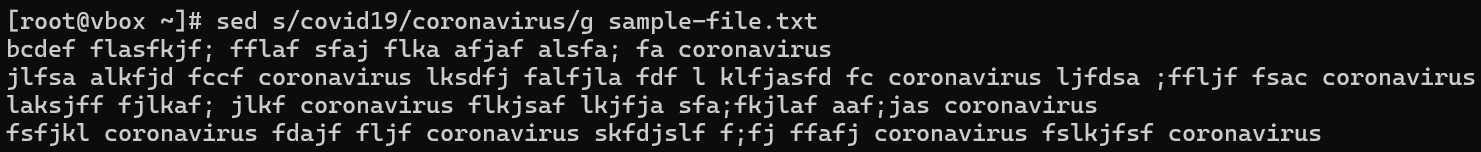
* 
* **whoami**: user name will be displayed
* **pwd**: present working directory
* **sudo -i:** switch to root user
* **#**: root user shell, **$**: normal user shell
* **/root:** home directory of root user, **/:** root directory (both are different)
  + **sudo -i**(from vagrant user’s shell)**:** takes to **/root** directory
  + **cd /:** takes to the root directory
* -----------------------------------------------------------------------------------------------------
* **touch:** create empty file
  + **touch <file\_name.extension\_name>**: create one file of the file name type
  + **touch myfile{1..5}.txt** : create 5 files I.e. myfile1.txt, .. myfile5.txt
* **mkdir:** create empty folder
  + **mkdir f1/f2/f3/f4/f5/f6**
    - If f1 or f2 is not exist, then it’ll not create all the folder
    - use **-p** flag in this case
    - 
* **cp <source> <destination>** : copy 
  + **cp mydevfile.txt dev/**
  + **cp mydevfiles{1..5}.txt dev/** : all 5 files will be copied
  + If you want to copy directories then use the flag -r
    - **cp -r <dir1> <destination>**
* **mv <source> <destination>** : move
  + No need to specify **-r**  here to move directories. You can directly move the directories here.
  + Ex: **mv /home/vagrant/devtemp2 /home/vagrant/dev**
  + **mv \*.txt testdir/** :move all txt files to testdir directory
* **rm <file\_path>** :delete a file
  + If you want to delete a directory: **rm -r <directory>**

**VIM EDITOR:**

* **vim <file\_name>** :create and open a file
  + 3 modes:
    - command mode: default, (after pressing escape key)
    - insert mode(edit mode): pressing **i** key
    - extended command mode: pressing **Esc** key then **:(colon)**key
      * w: write/save, q: quit
    - If you change something but don’t want to save it and quit,
      * **:q** will give error,
      * **:q!** will do the work.
      * **!** means **forcefully** do the things
* **:se nu**: set line numbers
  + 
  + To go to last line: **shift+g** or **G** (capslock on)
  + To go to first line: **gg** (small g+small g)
  + To copy the line: **yy**
    - If you want to copy multiple lines: **<number>yy** 
      * Ex: **4yy**: 4 lines will be copied starting from the line where cursor is present
  + To paste the copied line:
    - **p** (small p): paste below
    - **P** (capital P): paste above the cursor
  + To cut:
    - Same as copy
      * **dd**: cut
      * **4dd**: cut4 lines
  + To undo: **u**
  + To search any word: **/<keyword>:**Case-sensitive
    - To go to next keyword, presss “**n**”
  + **yy, p/P: copy paste, dd, p/P: cut paste**
* 
* 
* **Use this command to link a path. (just like shortcut app in windows)**
  + 
* To unlink the link:
  + **unlink <link>**
  + 
* To search any keyword:
  + **grep <keyword> <filepath>**
    - 
    - It is case sensitive, if you don’t want case sensitivity then mention **-i**.
    - To check inside a directory, use the flag **-R**
      * **(image in next page ------------->)**
      * 
    - **-v** keyword is used to see the files where the keyword is not present.
* **head** and **tail:**
  + **head:** first 10 lines of the file**, tail:** Last 10 lines of the file
    - **-f** is used to see if the logs are getting changed
    - **head -20 <file\_path>** : first 20 line you’ll be able to see.
* **Cut** command:
  + 
  + See this file, here in each line **:** is the separator present.
  + It’s just like the split function in javascript, that split each line with the separator.
  + **cut -d: -f4 /etc/passwd**
    - **-d:** =>separator is colon
    - **f4** => which field (in js index) you want to see after split (here **4**)
    - 
    - 
    - Here delimeter is required (-d)
* **awk** command:
  + 
  + **-F’<separator>**
  + **‘<command>’**
  + 
  + 
* 
* To replace any keyword in any file:
  + Method-1:
    - Open file using **vim**, in extended command mode write: **:%s/<current keyword>/<new keyword>** (note: this will replace only one word per line)
    - 
    - 
    - Here, coronavirus is the keyword to be replaced.
    - After executing that command, only first keyword of each line that is “coronavirus” will be replaced. If any more keyword is there in the same line, they won’t be replaced.
    - If you want to replace the words of first 3 lines only then:
      * **:1,3s/<old\_word>/<new\_word>/g**
    - **:%s/<current keyword>/<new keyword>/g** after adding **/g** all the keyword will be replaced.
    - **:%s/<current keyword>//g** if you want to replace it with nothing
  + **Method 2: (important)**
    - If there is multiple file that has to be changed, then method-1 will not be useful.
    - **sed** command will be used for this.
    - **sed ‘/s/<keyword>/<new\_keyword>/g’ <filepath>**
      * Here in case of **filepath** you can give **/\*.txt** kind of thing, every txt file will be change
      * 
      * Here it doesn’t change the orignial file, it will only display how the file will look like after replacing the keyword.
    - Without **-i** flag, it won’t replace. Correct command is:
      * **sed -i ‘/s/<keyword>/<new\_keyword>/g’ <filepath>**
* **Redirection: >**, **>>** (output redirection)and **<**, **<<** (input redirection)
  + Output redirection: transfer the output of a command to a file
  + Input redirection: get the file content of a file as input of a command.
  + If you don’t want to see the output of any command in the screen and you want to transfter the output to another file, then it is used.
  + **ls > <filepath>**  => it will transfer the output that ls would have generated to the file which path is mentioned.
    - If the file doesn’t exist, it’ll create one and transfer the content.
    - If the file exists, it’ll override the file.
  + **ls >> <filepath>** => It’ll not override, rather it’ll append the content
  + 
  + >> means **1>>** (default behaviour) means **output redirection**
  + If you want **error redirection** then **2>>**
    - 
    - ***&>> is for both Output and Error redirection.***
    - There is a file **/dev/null**, which is empty. If you redirect anything to this file-path, the output will neither be generated in the terminal nor in this file. If you redirect the content of this file to another file then the new will won’t have any content.
  + 
    - (line words characters file-path) output of **wc** command
    - In first command: **wc** (word count) command gets the file as input, so it has the knowledge about the file. So it prints the filename
    - In second command: **wc** command gets the content of the file as input, so it doesn’t know about the filename.
    - This is the use of input redirection.
* **Pipe: |**
  + **wc** => used to count something in a file. (**-l** to count number of lines)
    - 
    - 
      * Lines, words, characters, filepath
    - 
    - Symbol of Pipe is: **|**
    - 
      * Here **ls** generate the list of the file names inside the etc directory.
      * This will be used as the input of **wc -l**
      * **wc -l** takes the file content (in case of input redirection) and gives the number of lines, so think output of **ls** as the file content.
    - 
      * This is the same as the previous pipe example.
  + 
  +  (both same)
    - * Search all the files that contains **host** keyword in it.
* **Find:**
  + It is used to find any file
  + 
* **Users and Groups:**
  + 
  + 
  + **username:password:UID:GID:GECOS:home\_directory:shell**
    - **GECOS:** extra info about the user like full name or something like that.
  + **john:x:1001:1001:John Doe:/home/john:/bin/bash**
    - **x:** link to the password file
  + **/etc/passwd :passwd file, /etc/group : group file**
  + 
    - **id** command is used to see the user info.
  + **useradd <username>** => create new user and add it to /etc/passwd file
    - 
  + **useradd --home-dir /opt/tomcat --shell /sbin/nologin tomcat**
    - It means /opt/tomcat will be the home directory.
    - 
  + **groupadd** =>
    - 
    - **devops** group was created and ansible was added to this group.
    - **G** flag is for secondary group, **g** flag is for primary group.
    - ansible is now inside the devops group (last line of image)
    - **usermod** is used to add the user in a group.
    - **Otherwise open the */etc/group* file and write the users name (comma separated)**
    - 
    - 
* **-rw-r--r-- 1 john developers 1234 Mar 16 14:00 file.txt**
  + This is the output of a file after doing: **ls -l <file\_path>**
  + **rw:** (1st one)owner’s permission
  + **r:** (2nd one)group’s permission
  + **r:** (3rd one)**other’s permission**
* **File permissions:**
  + 
  + **drwxr-xrw-** total 10 characters
    - First char: file type (here d: directory)
    - Then 3 chars 3 times (rwx) format
      * **Rwx :** root user’s permissions
      * **r-x** : read, execute (not write)
      * **rw-** : read, write (not execute)
* **Change Ownership of a File:** 
  + **chown** command can be used for this
    - **chown -R <user name>:<group name> <directory path>** (If u give -R, it’ll change the ownership of all the sub-directories as well)
    - Or: **chown -R <user name>.<group name> <directory path>**
  + **chmod** is used to change the access like read, write,
    - **chmod o-x <directory path>** (it’ll revoke permission for “others” to execute) (“o” for others, “-” means revoke, “+” means give permission, “g” for group, “u” for user)
    - Numeric values for
      * R = 4
      * Write = 2
      * Execute = 1
    - If u want to give full access to user, group and no permission to others then
      * chmod -R 770 <directory> (7 = 4 + 2 + 1)
* **Sudo:**
  + You can create any new file inside the **/etc/sudoers.d** directory to give sudo access to any user.
  + It is not recommended to use **/etc/sudoers** file.
* **Packages:**
  + To see the repositories of yum: **/etc/yum.repos.d/**
  + To download: **wget**, **curl** (curl has also some different usability)
  + **curl <url> -o <path name of the would be file>** (**-o** means output)
  + After downloading, it can be installed using the following command:
    - **rpm -ivh <file path>** (i: install, v: verbose i.e. output, h: human readable format)
  + **yum search <package name>** (to search any package, it’ll install all the dependent package)
  + Now a modern package manager is there in **rpm** based Linux: **dnf**
  + In place of yum, we should use **dnf**
  + **dnf repolist**: it’ll give all the repos that have in there in the system.
* **Services:**
  + **systemctl**: to operate the service (for ex: **httpd** is a service that we installed before)…
    - systemctl start <service>
    - systemctl stop <service> …etc etc
* **Processes:**
  + **ps -ef**: show all the processes
  + **kill <process id>**  (to kill any process, if any children processes are there then it’ll stop them and then it’ll stop itself)
  + **kill -9 <process id>** (to stop the process forcefully)
* **xargs** converts piped input into valid command-line arguments.
  + - **echo "file1.txt file2.txt" | rm**
      * It’ll not work as rm doesn’t expect piped input, it needs arguments in proper format like **rm file1.txt file2.txt**
      * **echo “file1.txt file2.txt” | *xargs* rm** 
        + It’ll work properly
  + 
    - name="hello"
    - awk "{ print **$name** }" file.txt
    - It is same as awk "{print hello}" file.txt
    - Here, awk '{ print $name }' doesn’t interprete the value of name variable.
    - '……' => Prevents shell from interpreting $, \, etc. Best for awk code.
    - ".……" => Shell does interpret variables inside. Avoid for awk code unless needed.
* **Archiving**
  + 
    - Zip
  + 
    - Unzip
  + Another command that is used is: **zip** 
    - It might not be there so install it using: **yum install zip**
    - 
* **Ubuntu Commands**
  + 
    - Unlike **centos**, in **ubuntu**, when we create one user it doesn’t create the home directory for that user.
  + 
    - Use **adduser** command in place of **useradd**.
  + **visudo**
    - When you give this command, it’ll open the **/etc/sudoers** file in **nano** editor.
    - To open it in vim editor:
      * 
  + 
    - Downloaded one package named “tree”.
    - 
  + **dpkg -l**
    - To list all debian packages in the machine
  + *Unline* ***yum*** *in centos, in Ubuntu* ***apt*** *is there.*
  + 
  + **apt install <package name>**
    - It’ll install the package.
    - In ubuntu, when a package it installed, it automatically get started and enabled.
  + **apt remove <package name>**
    - Removes the package but will *not properly remove* all the configs and all.
  + **apt purge <package name>**
    - Remove the packages as well as the configurations.
  + **sudo**:
    - Runs a command as another user (default: root) but stay as your original user.
    - Needs **own password** (not root’s)
    - Requires you to be in the **sudoers** group.
  + **su**
    - Switches to another user account (default: root) and gives you an interactive shell
    - Needs the **target user’s password**
    - No logging or restriction like **sudo**.
    - Ex: **su john** (requires john’s password)
  + **-u <username>**
    - Used with **sudo** to specify a user to run the command as.
    - Ex: **sudo -u xyz <some command>** : It’ll run the command as user **xyz**.
  + **sudo -i :** 
    - It checks whether the user is allowed to execute commands as root in **/etc/sudoers.d** directory. If he is allowed then only he can execute.
    - You might be seeing, when you give command ***sudo -i*** in vagrant default user, it doesn’t ask for the password. It is because it is mentioned in the ***/etc/sudoers.d/vagrant***
    - 
    - 
      * If it is not mentioned, you’ll see this kind of one error.
  + **su**
    - When you give **su**, it switches to the root user (default: root), so to authorize you to be the root user, it needs the ***root user password***.
    - 
      * See, here to switch as the root user, you need the root user’s password. If you are giving correct password, you can switch to the root user.
  + In short: **su** changes the current user to a specific user (su john: means the user is john.. if no user is mentioned then root user) which requires perticular user’s password. Whereas **sudo -i** allows you to execute the commands as root user (if you are allowed means if you are being allowed in the /etc/sudoers.d/ directory.
  + If you want to open the shell of any perticular user
    - ***sudo -i -u <username>*** or ***sudo -u <username> -i*** (2nd one is better understandable)
    - It’s kind of running **-i** command as the user **<username>**
    - **sudo -i** is shorthand for **sudo -u root -i**.
    - 
* **Writing into a file directly from the command line:**
  + **cat >** **hello.txt** (if it is not there, then it’ll create one)
    - Write whatever you want… then press **Ctrl+D**
    - Now, the texts you entered will be overwritten in side the hello.txt
    - If you want to append then
      * **cat >> hello.txt**
  + If you want to add the content along with the command itself then:
    - *printf “alok* ***\n****ranjan* ***\n****joshi” > hello.txt (I used \n to add multilines)*
    - *Or echo “something something” > hello.txt*
      * 
        + Anything that comes under **<<EOF** and **EOF** will be saved in side that file. (Note: here <<EOF is not any input redirection)
      * 
        + This one is looking better.
* 
* When you are downloading any repo (extra repo using ***dnf*** or something) it might be disabled by default. I mean when you execute ***dnf install <some package>***  it’ll check in the repo that are enable. So, if your package is not in the enabled repo list, then it won’t get downloaded. So, to download your package, your repo must be enabled.
  + You can either permanently enable the repo by editing the ***.repo*** file: ***enabled=1***
  + Else, you can temporarily enable the repo:
    - ***dnf --enablerepo=<repo-name> install <package>***
* ***NOTE***
* ***sudo echo “something” > /root/testfile.txt***
  + It’ll give error as *echo* has been executed as super-user (sudo) but output redirection is executed by current user only.
  + So use ***sh*** to run the whole command string inside a new shell having root user privileges.
  + ***sudo sh -c “something” > /root/testfile.txt***
* **Some Examples:**
  + **head and tail:**
    - 
    - 
  + **cut**
    - **Splitting the file with delimeter.**
    - 
    - **Or**
    - 
    - **Getting the values at any positions:**
    - 
  + **awk**
    - 
    - You can give **-F’:’** as well.
    - 
    - 
    - ***awk*** command is very useful to query a file, here I extracted all 2nd indexed values after splitting with \_ (which are 1, 2, ..50) and printing out their sum.
  + **Sed**
    - **To see the file replacing the keyword (not replacing in the original file)**
    - 
    - Use **-i** flag to replace in the original file.
    - 