**Linux Commands in DevOps**

|  |  |
| --- | --- |
| ***ls <path name>*** | By specifying the path after ls, the content in that path will be displayed |
| ***ls –l*** | Using ‘l’ flag, lists all the contents along with its owner settings, permissions & time  stamp (long format) |
| ***ls –a*** | Using ‘a’ flag, lists all the hidden contents in the specified directory |

**2. sudo**

This command executes only that command with root/ super user privileges.

*$ sudo <command>*

|  |  |
| --- | --- |
| **Command** | **Description** |
| ***sudo useradd <username>*** | Adding a new user |
| ***sudo passwd <username>*** | Setting a password for the new user |
| ***sudo userdel <username>*** | Deleting the user |
| ***sudo groupadd <groupname>*** | Adding a new group |
| ***sudo groupdel <groupname>*** | Deleting the  group |
| ***sudo usermod -g <groupname> <username>*** | Adding a user to a primary group |

**3. Cat**

This command can read, modify or concatenate text files. It also displays file contents.

*$ cat <flag> {filename}*

|  |  |
| --- | --- |
| Command | Description |
| ***cat -b*** | This adds line numbers to non-blank lines |
| ***cat -n*** | This adds line numbers to all lines |
| ***cat -s*** | This squeezes blank lines into one line |
| ***cat –E*** | This shows $ at the end of line |

**4. grep**

This command searches for a particular string/ word in a text file. This is similar to “Ctrl+F” but executed via a CLI.

**syntax:**

*$ grep <flag or element\_to\_search> {filename}*

|  |  |
| --- | --- |
| Command | Description |
| ***grep -i*** | Returns the results for case insensitive strings |
| ***grep -n*** | Returns the matching strings along with their line number |
| ***grep -v*** | Returns the result of lines not matching the search string |
| ***grep -c*** | Returns the number of lines in which the results matched the search string |

**5. sort**

This command sorts the results of a search either alphabetically or numerically. It also sorts files, file contents, and directories.

***syntax:***

*$ sort <flag> {filename}*

|  |  |
| --- | --- |
| Command | Description |
| ***sort -r*** | the flag returns the results in reverse order; |
| ***sort -f*** | the flag does case insensitive sorting |
| ***sort -n*** | the flag returns the results as per numerical order |

**6.tail**

It is complementary to head command. The tail command, as the name implies, print the last N number of data of the given input. By default, it prints the last 10 lines of the specified files. If you give more than one filename, then data from each file precedes by its file name.

**syntax:**

tail [OPTION]... [FILE]...

**tail -n 3 state.txt or tail -3 state.txt => -n for no. of lines**

tail +25 state.txt

7. **chown**

Different users in the operating system have ownership and permission to ensure that the files are secure and put restrictions on who can modify the contents of the files. In Linux there are different users who use the system:

* Each *user*has some properties associated with them, such as a user ID and a home directory. We can add users into a group to make the process of managing users easier.
* A *group*can have zero or more users. A specified user is associated with a “default group”. It can also be a member of other groups on the system as well.

**Ownership and Permissions:** To protect and secure files and directory in Linux we use permissions to control what a user can do with a file or directory. Linux uses three types of permissions:

* **Read:**This permission allows the user to read files and in directories, it lets the user read directories and subdirectories stores in it.
* **Write:**This permission allows a user to modify and delete a file. Also, it allows a user to modify its contents (create, delete and rename files in it) for the directories. Unless you give the execute permission to directories, changes does not affect them.
* **Execute:**The write permission on a file executes the file. For example, if we have a file named *sh* so unless we don’t give it execute permission it won’t run.

**Types of file Permissions:**

* **User:**This type of file permission affects the owner of the file.
* **Group:**This type of file permission affects the group which owns the file. Instead of the group permissions, the user permissions will apply if the owner user is in this group.
* **Other: This** type of file permission affects all other users on the system.

**Note:** To view the permissions we use:

*ls -l*

**chown** command is used to change the file Owner or group. Whenever you want to change ownership you can use chown command.

**Syntax:**

*chown [OPTION]… [OWNER][:[GROUP]] FILE…*

*chown [OPTION]… –reference=RFILE FILE…*

**Example:** To change owner of the file:

*chown owner\_name file\_name*

*chown master file1.txt*

where the *master*is another user in the system. Assume that if you are user named user1 and you want to change ownership to root (where your current directory is user1). use “sudo” before syntax.

*sudo chown root file1.txt*

**chmod**

This command is used to change the access permissions of files and directories.

Syntax:

*chmod <permissions of user,group,others> {filename}*

4 – read permission

2 – write permission

1 – execute permission

0 – no permission

**8.lsof**

While working in Linux/Unix system there might be several file and folder which are being used, some of them would be visible and some not. **lsof** command stands for **List Of Open File**. This command provides a list of files that are opened. Basically, it gives the information to find out the files which are opened by which process. With one go it lists out all open files in the output console.

**Syntax:**

*$lsof [option][user name]*

**Options with Examples:**

* **List all open files:**This command lists out all the files that are opened by any process in the system.

*~$ lsof*

* Here, you observe there are details of the opened files. ProcessId, the user associated with the process, FD(file descriptor), size of the file all together gives detailed information about the file opened by the command, process ID, user, its size, etc.
* **FD**represents as File descriptor.
* **cwd**: Current working directory.
* **txt:** Text file.
* **mem**: Memory file.
* **mmap**: Memory-mapped device.

**List all files opened by a user:**There are several users of a system and each user has different requirements and accordingly they use files and devices. To find a list of files that are opened by a specific user this command is useful.

* **Syntax:**
* *lsof -u username*

Along with that we can see the type of file here and they are:

* **DIR:**Directory
* **REG:**Regular file
* **CHR:**Character special file

9. **ifconfig**

**ifconfig**(interface configuration) command is used to configure the kernel-resident network interfaces. It is used at the boot time to set up the interfaces as necessary. After that, it is usually used when needed during debugging or when you need system tuning. Also, this command is used to assign the IP address and netmask to an interface or to enable or disable a given interface.

**Syntax:**

*ifconfig [...OPTIONS] [INTERFACE]*

**Options:**

* **-a :**This option is used to display all the interfaces available, even if they are down.

**Syntax:**

*ifconfig -a*

**-s :**Display a short list, instead of details.

**Syntax:**

*ifconfig -s*

10. **history**

historycommand is used to view the previously executed command. This feature was not available in the Bourne shell. Bash and Korn support this feature in which every command executed is treated as the event and is associated with an event number using which they can be recalled and changed if required. These commands are saved in a history file. In Bash shell **history** command shows the whole list of the command.

**Syntax:**

*$ history*

To show the limited number of commands that executed previously as follows:

*$ history 10*

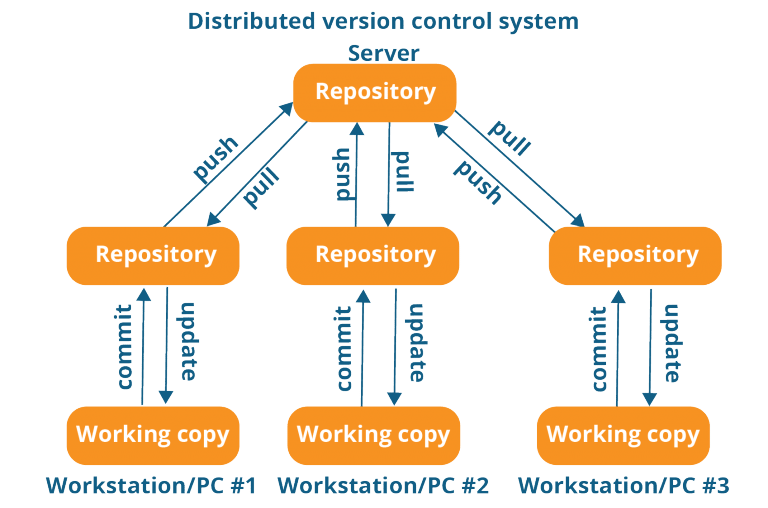
11. **ssh-keygen**

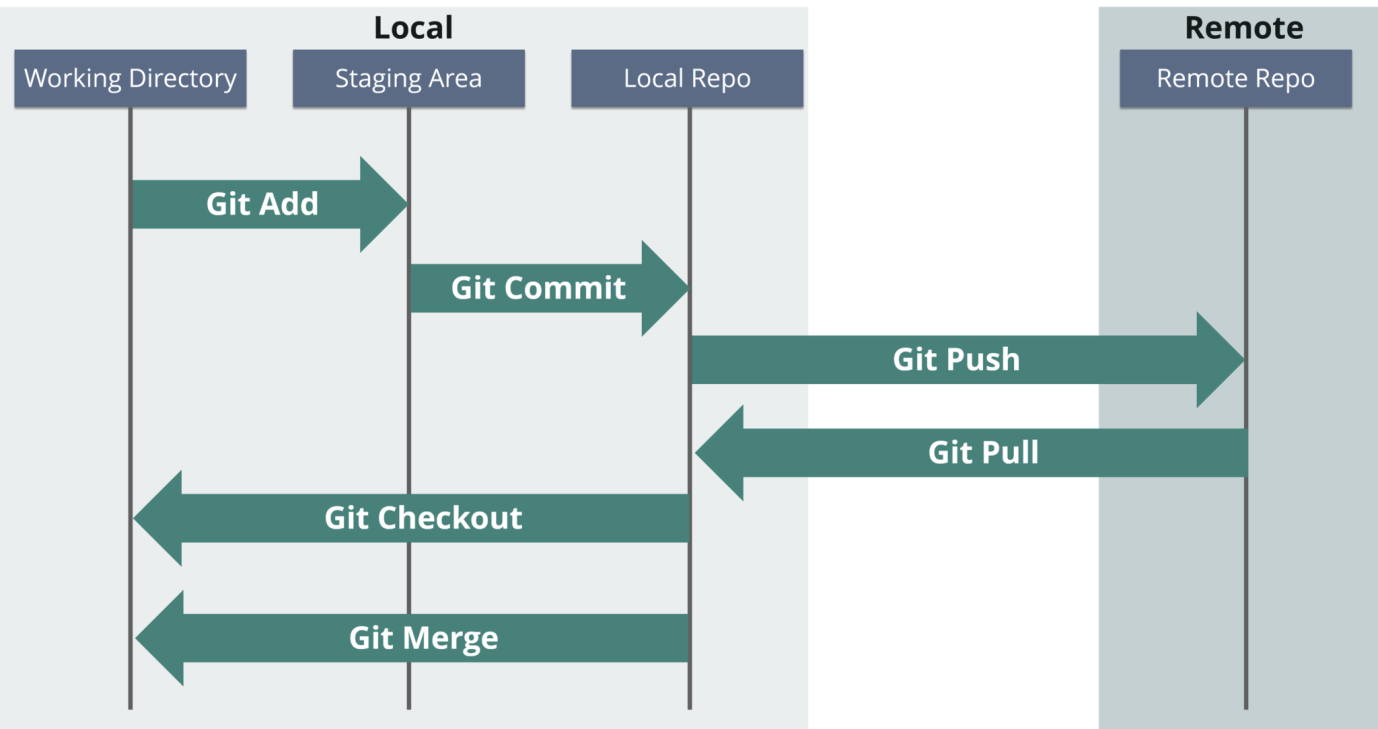
Use the ssh-keygen command to generate a public/private authentication key pair. Authentication keys allow a user to connect to a remote system without supplying a password. Keys must be generated for each user separately. If you generate key pairs as the root user, only the root can use the keys.

The following example creates the public and private parts of an RSA key:

*ssh-keygen -t rsa*

## ****Git Commands****





**1.git init**

**Usage**: git init [repository name]

This command creates a new repository.

2. **git config**

**Usage**:*git config --global user.name “[name]”*

**Usage**: *git config --global user.email “[email address]”*

This command sets the author name and email address respectively. This is useful information with the commits.

3. **git clone**

**Usage**:*git clone [url]*

This command lets you get a copy of a repository from local .

4. **git add**

**Usage:***git add [file] or*

This command adds a file to the staging area.

**5.git commit**

**Usage:** *git commit -m “[ Type in the commit message]”*

This command records or snapshots the file permanently in the version history.

**Usage:** *git commit -a*

This command commits any files you’ve added with the git add command and also commits any files you’ve changed since then.

6. **git status**

**Usage:***git status*

The git status command displays the state of the working directory and the staging area.This command lets you see the changes that are in the staging, those that are not staged and are not tracked by Git.

7. **git show**

**Usage:** *git show*

This command shows the metadata and content changes of the specified commit.

8. **git rm**

**Usage:** *git rm [file]*

This command deletes the file from your working directory and stages the deletion.

9. **git remote**

**Usage:** *git remote add [variable name] [Remote Server Link]*

This command connects your local repository to the remote server.

10. **git push**

**Usage:** *git push [variable name] master*

**Usage:** *git push [variable name] [branch]*

This command sends the branch commits to your remote repository.

**Usage:** *git push –all [variable name]*

This command pushes all branches to your remote repository.

**Usage:** *git push [variable name] :[branch name]*

This command deletes a branch on your remote repository.

11. **git pull**

**Usage:** *git pull [Repository Link]*

This command fetches and merges changes on the remote server to your working directory.

12. **git branch**

**Usage:***git branch*

This command lists all the local branches in the current repository.

**Usage:** *git branch [branch name]*

This command creates a new branch.

**Usage:** *git branch -d [branch name]*

This command deletes the feature branch.

13. **git checkout**

**Usage:** *git checkout [branch name]*

This command lets you switch from one branch to another.

**Usage:***git checkout -b [branch name]*

This command creates a new branch and also switches to it.

14. **git merge**

**Usage:** *git merge [branch name]*

This command merges the specified branch’s history into the current branch.

15. **git rebase**

**Usage:** *git rebase [branch name]*

*git rebase master* – This command will move all our work from the current branch to the master.