DevOps Commands Cheat Sheet

Basic Linux Commands -

Linux is the foundation of DevOps operations - it's like a Swiss Army knife for servers. These commands help you navigate systems, manage files, configure permissions, and automate tasks in terminal environments.

1. **pwd** - Print the current working directory.
2. **ls** - List files and directories.
3. **cd** - Change directory.
4. **touch** - Create an empty file.
5. **mkdir** - Create a new directory.
6. **rm** - Remove files or directories.
7. **rmdir** - Remove empty directories.
8. **cp** - Copy files or directories.
9. **mv** - Move or rename files and directories.
10. **cat** - Display the content of a file.
11. **echo** - Display a line of text.
12. **clear** - Clear the terminal screen.

Intermediate Linux Commands

13. **chmod** - Change file permissions.
14. **chown** - Change file ownership.
15. **find** - Search for files and directories.
16. **grep** - Search for text in a file.
17. **wc** - Count lines, words, and characters in a file.
18. **head** - Display the first few lines of a file.
19. **tail** - Display the last few lines of a file.
20. **sort** - Sort the contents of a file.
21. **uniq** - Remove duplicate lines from a file.
22. **diff** - Compare two files line by line.
23. **tar** - Archive files into a tarball.
24. **zip/unzip** - Compress and extract ZIP files.
25. **df** - Display disk space usage.
26. **du** - Display directory size.
27. **top** - Monitor system processes in real time.
28. **ps** - Display active processes.
29. **kill** - Terminate a process by its PID.
30. **ping** - Check network connectivity.
31. **wget** - Download files from the internet.
32. **curl** - Transfer data from or to a server.
33. **scp** - Securely copy files between systems.
34. **rsync** - Synchronize files and directories.

Advanced Linux Commands

35. **awk** - Text processing and pattern scanning.
36. **sed** - Stream editor for filtering and transforming text.
37. **cut** - Remove sections from each line of a file.
38. **tr** - Translate or delete characters.
39. **xargs** - Build and execute command lines from standard input.
40. **ln** - Create symbolic or hard links.
41. **df -h** - Display disk usage in human-readable format.
42. **free** - Display memory usage.
43. **iostat** - Display CPU and I/O statistics.
44. **netstat** - Network statistics (use `ss` as modern alternative).
45. **ifconfig/ip** - Configure network interfaces (use `ip` as modern alternative).
46. **iptables** - Configure firewall rules.
47. **systemctl** - Control the systemd system and service manager.
48. **journalctl** - View system logs.
49. **crontab** - Schedule recurring tasks.
50. **at** - Schedule tasks for a specific time.
51. **uptime** - Display system uptime.
52. **whoami** - Display the current user.
53. **users** - List all users currently logged in.
54. **hostname** - Display or set the system hostname.
55. **env** - Display environment variables.
56. **export** - Set environment variables.

Networking Commands

57. **ip addr** - Display or configure IP addresses.
58. **ip route** - Show or manipulate routing tables.
59. **traceroute** - Trace the route packets take to a host.
60. **nslookup** - Query DNS records.
61. **dig** - Query DNS servers.
62. **ssh** - Connect to a remote server via SSH.
63. **ftp** - Transfer files using the FTP protocol.
64. **nmap** - Network scanning and discovery.
65. **telnet** - Communicate with remote hosts.
66. **netcat (nc)** - Read/write data over networks.

File Management and Search

67. **locate** - Find files quickly using a database.
68. **stat** - Display detailed information about a file.

69. **tree** - Display directories as a tree.
70. **file** - Determine a file's type.
71. **basename** - Extract the filename from a path.
72. **dirname** - Extract the directory part of a path.

System Monitoring

73. **vmstat** - Display virtual memory statistics.
74. **htop** - Interactive process viewer (alternative to `top`).
75. **lsof** - List open files.
76. **dmesg** - Print kernel ring buffer messages.
77. **uptime** - Show how long the system has been running.
78. **iotop** - Display real-time disk I/O by processes.

Package Management

79. **apt** - Package manager for Debian-based distributions.
80. **yum/dnf** - Package manager for RHEL-based distributions.
81. **snap** - Manage snap packages.
82. **rpm** - Manage RPM packages.

Disk and Filesystem

83. **mount/umount** - Mount or unmount filesystems.
84. **fsck** - Check and repair filesystems.
85. **mkfs** - Create a new filesystem.
86. **blkid** - Display information about block devices.
87. **lsblk** - List information about block devices.
88. **parted** - Manage partitions interactively.

Scripting and Automation

89. **bash** - Command interpreter and scripting shell.
90. **sh** - Legacy shell interpreter.
91. **cron** - Automate tasks.
92. **alias** - Create shortcuts for commands.
93. **source** - Execute commands from a file in the current shell.

Development and Debugging

94. **gcc** - Compile C programs.
95. **make** - Build and manage projects.
96. **strace** - Trace system calls and signals.
97. **gdb** - Debug programs.
98. **git** - Version control system.
99. **vim/nano** - Text editors for scripting and editing.

Other Useful Commands

100. **uptime** - Display system uptime.
101. **date** - Display or set the system date and time.
102. **cal** - Display a calendar.
103. **man** - Display the manual for a command.
104. **history** - Show previously executed commands.
105. **alias** - Create custom shortcuts for commands.

Basic Git Commands

Git is your code time machine. It tracks every change, enables team collaboration without conflicts, and lets you undo mistakes. These commands help manage source code versions like a professional developer.

1. **git init**
   Initializes a new Git repository in the current directory.
   Example: `git init`
2. **git clone**
   Copies a remote repository to the local machine.
   Example: `git clone https://github.com/user/repo.git`
3. **git status**
   Displays the state of the working directory and staging area.
   Example: `git status`
4. **git add**
   Adds changes to the staging area.
   Example: `git add file.txt`
5. **git commit**
   Records changes to the repository.
   Example: `git commit -m "Initial commit"`
6. **git config**
   Configures user settings, such as name and email.
   Example: `git config --global user.name "Your Name"`

7. **git log**
   Shows the commit history.
   Example: `git log`
8. **git show**
   Displays detailed information about a specific commit.
   Example: `git show <commit-hash>`
9. **git diff**
   Shows changes between commits, the working directory, and the staging area.
   Example: `git diff`
10. **git reset**
    Unstages changes or resets commits.
    Example: `git reset HEAD file.txt`

Branching and Merging

11. **git branch**
    Lists branches or creates a new branch.
    Example: `git branch feature-branch`
12. **git checkout**
    Switches between branches or restores files.
    Example: `git checkout feature-branch`
13. **git switch**
    Switches branches (modern alternative to `git checkout`).
    Example: `git switch feature-branch`
14. **git merge**
    Combines changes from one branch into another.
    Example: `git merge feature-branch`
15. **git rebase**
    Moves or combines commits from one branch onto another.
    Example: `git rebase main`
16. **git cherry-pick**
    Applies specific commits from one branch to another.
    Example: `git cherry-pick <commit-hash>`

Remote Repositories

17. **git remote**
    Manages remote repository connections.
    Example: `git remote add origin https://github.com/user/repo.git`
18. **git push**
    Sends changes to a remote repository.
    Example: `git push origin main`

19. **git pull**
    Fetches and merges changes from a remote repository.
    Example: `git pull origin main`
20. **git fetch**
    Downloads changes from a remote repository without merging.
    Example: `git fetch origin`
21. **git remote -v**
    Lists the URLs of remote repositories.
    Example: `git remote -v`

Stashing and Cleaning

22. **git stash**
    Temporarily saves changes not yet committed.
    Example: `git stash`
23. **git stash pop**
    Applies stashed changes and removes them from the stash list.
    Example: `git stash pop`
24. **git stash list**
    Lists all stashes.
    Example: `git stash list`
25. **git clean**
    Removes untracked files from the working directory.
    Example: `git clean -f`

Tagging

26. **git tag**
    Creates a tag for a specific commit.
    Example: `git tag -a v1.0 -m "Version 1.0"`
27. **git tag -d**
    Deletes a tag.
    Example: `git tag -d v1.0`
28. **git push --tags**
    Pushes tags to a remote repository.
    Example: `git push origin --tags`

Advanced Commands

29. **git bisect**
    Finds the commit that introduced a bug.
    Example: `git bisect start`
30. **git blame**
    Shows which commit and author modified each line of a file.
    Example: `git blame file.txt`
31. **git reflog**
    Shows a log of changes to the tip of branches.
    Example: `git reflog`
32. **git submodule**
    Manages external repositories as submodules.
    Example: `git submodule add https://github.com/user/repo.git`
33. **git archive**
    Creates an archive of the repository files.
    Example: `git archive --format=zip HEAD > archive.zip`
34. **git gc**
    Cleans up unnecessary files and optimizes the repository.
    Example: `git gc`

GitHub-Specific Commands

35. **gh auth login**
    Logs into GitHub via the command line.
    Example: `gh auth login`
36. **gh repo clone**
    Clones a GitHub repository.
    Example: `gh repo clone user/repo`
37. **gh issue list**
    Lists issues in a GitHub repository.
    Example: `gh issue list`
38. **gh pr create**
    Creates a pull request on GitHub.
    Example: `gh pr create --title "New Feature" --body "Description of the feature"`
39. **gh repo create**
    Creates a new GitHub repository.
    Example: `gh repo create my-repo`

Basic Docker Commands -

Docker packages applications into portable containers - like shipping containers for software. These commands help build, ship, and run applications consistently across any environment.

1. **docker --version**
   Displays the installed Docker version.
   Example: `docker --version`
2. **docker info**
   Shows system-wide information about Docker, such as the number of containers and images.
   Example: `docker info`
3. **docker pull**
   Downloads an image from a Docker registry (default: Docker Hub).
   Example: `docker pull ubuntu:latest`
4. **docker images**
   Lists all downloaded images.
   Example: `docker images`
5. **docker run**
   Creates and starts a new container from an image.
   Example: `docker run -it ubuntu bash`
6. **docker ps**
   Lists running containers.
   Example: `docker ps`
7. **docker ps -a**
   Lists all containers, including stopped ones.
   Example: `docker ps -a`
8. **docker stop**
   Stops a running container.
   Example: `docker stop container_name`
9. **docker start**
   Starts a stopped container.
   Example: `docker start container_name`
10. **docker rm**
    Removes a container.
    Example: `docker rm container_name`
11. **docker rmi**
    Removes an image.
    Example: `docker rmi image_name`
12. **docker exec**
    Runs a command inside a running container.
    Example: `docker exec -it container_name bash`

Intermediate Docker Commands

13. **docker build**
    Builds an image from a Dockerfile.
    Example: `docker build -t my_image .`
14. **docker commit**
    Creates a new image from a container's changes.
    Example: `docker commit container_name my_image:tag`
15. **docker logs**
    Fetches logs from a container.
    Example: `docker logs container_name`
16. **docker inspect**
    Returns detailed information about an object (container or image).
    Example: `docker inspect container_name`
17. **docker stats**
    Displays live resource usage statistics of running containers.
    Example: `docker stats`
18. **docker cp**
    Copies files between a container and the host.
    Example: `docker cp container_name:/path/in/container /path/on/host`
19. **docker rename**
    Renames a container.
    Example: `docker rename old_name new_name`
20. **docker network ls**
    Lists all Docker networks.
    Example: `docker network ls`
21. **docker network create**
    Creates a new Docker network.
    Example: `docker network create my_network`
22. **docker network inspect**
    Shows details about a Docker network.
    Example: `docker network inspect my_network`
23. **docker network connect**
    Connects a container to a network.
    Example: `docker network connect my_network container_name`
24. **docker volume ls**
    Lists all Docker volumes.
    Example: `docker volume ls`
25. **docker volume create**
    Creates a new Docker volume.
    Example: `docker volume create my_volume`
26. **docker volume inspect**
    Provides details about a volume.
    Example: `docker volume inspect my_volume`

27. **docker volume rm**
    Removes a Docker volume.
    Example: `docker volume rm my_volume`

Advanced Docker Commands

28. **docker-compose up**
    Starts services defined in a `docker-compose.yml` file.
    Example: `docker-compose up`
29. **docker-compose down**
    Stops and removes services defined in a `docker-compose.yml` file.
    Example: `docker-compose down`
30. **docker-compose logs**
    Displays logs for services managed by Docker Compose.
    Example: `docker-compose logs`
31. **docker-compose exec**
    Runs a command in a service's container.
    Example: `docker-compose exec service_name bash`
32. **docker save**
    Exports an image to a tar file.
    Example: `docker save -o my_image.tar my_image:tag`
33. **docker load**
    Imports an image from a tar file.
    Example: `docker load < my_image.tar`
34. **docker export**
    Exports a container's filesystem as a tar file.
    Example: `docker export container_name > container.tar`
35. **docker import**
    Creates an image from an exported container.
    Example: `docker import container.tar my_new_image`
36. **docker system df**
    Displays disk usage by Docker objects.
    Example: `docker system df`
37. **docker system prune**
    Cleans up unused Docker resources (images, containers, volumes, networks).
    Example: `docker system prune`
38. **docker tag**
    Assigns a new tag to an image.
    Example: `docker tag old_image_name new_image_name`
39. **docker push**
    Uploads an image to a Docker registry.
    Example: `docker push my_image:tag`

40. **docker login**
    Logs into a Docker registry.
    Example: `docker login`
41. **docker logout**
    Logs out of a Docker registry.
    Example: `docker logout`
42. **docker swarm init**
    Initializes a Docker Swarm mode cluster.
    Example: `docker swarm init`
43. **docker service create**
    Creates a new service in Swarm mode.
    Example: `docker service create --name my_service nginx`
44. **docker stack deploy**
    Deploys a stack using a Compose file in Swarm mode.
    Example: `docker stack deploy -c docker-compose.yml my_stack`
45. **docker stack rm**
    Removes a stack in Swarm mode.
    Example: `docker stack rm my_stack`
46. **docker checkpoint create**
    Creates a checkpoint for a container.
    Example: `docker checkpoint create container_name checkpoint_name`
47. **docker checkpoint ls**
    Lists checkpoints for a container.
    Example: `docker checkpoint ls container_name`
48. **docker checkpoint rm**
    Removes a checkpoint.
    Example: `docker checkpoint rm container_name checkpoint_name`

Basic Kubernetes Commands -

Kubernetes is the conductor of your container orchestra. It automates deployment, scaling, and management of containerized applications across server clusters.

1. **kubectl version**
   Displays the Kubernetes client and server version.
   Example: `kubectl version --short`
2. **kubectl cluster-info**
   Shows information about the Kubernetes cluster.
   Example: `kubectl cluster-info`
3. **kubectl get nodes**
   Lists all nodes in the cluster.
   Example: `kubectl get nodes`
4. **kubectl get pods**
   Lists all pods in the default namespace.
   Example: `kubectl get pods`

5. **kubectl get services**
   Lists all services in the default namespace.
   Example: `kubectl get services`
6. **kubectl get namespaces**
   Lists all namespaces in the cluster.
   Example: `kubectl get namespaces`
7. **kubectl describe pod**
   Shows detailed information about a specific pod.
   Example: `kubectl describe pod pod-name`
8. **kubectl logs**
   Displays logs for a specific pod.
   Example: `kubectl logs pod-name`
9. **kubectl create namespace**
   Creates a new namespace.
   Example: `kubectl create namespace my-namespace`
10. **kubectl delete pod**
    Deletes a specific pod.
    Example: `kubectl delete pod pod-name`

Intermediate Kubernetes Commands

11. **kubectl apply**
    Applies changes defined in a YAML file.
    Example: `kubectl apply -f deployment.yaml`
12. **kubectl delete**
    Deletes resources defined in a YAML file.
    Example: `kubectl delete -f deployment.yaml`
13. **kubectl scale**
    Scales a deployment to the desired number of replicas.
    Example: `kubectl scale deployment my-deployment --replicas=3`
14. **kubectl expose**
    Exposes a pod or deployment as a service.
    Example: `kubectl expose deployment my-deployment --type=LoadBalancer --port=80`
15. **kubectl exec**
    Executes a command in a running pod.
    Example: `kubectl exec -it pod-name -- /bin/bash`
16. **kubectl port-forward**
    Forwards a local port to a port in a pod.
    Example: `kubectl port-forward pod-name 8080:80`
17. **kubectl get configmaps**
    Lists all ConfigMaps in the namespace.
    Example: `kubectl get configmaps`

18. **kubectl get secrets**
    Lists all Secrets in the namespace.
    Example: `kubectl get secrets`
19. **kubectl edit**
    Edits a resource definition directly in the editor.
    Example: `kubectl edit deployment my-deployment`
20. **kubectl rollout status**
    Displays the status of a deployment rollout.
    Example: `kubectl rollout status deployment/my-deployment`

Advanced Kubernetes Commands

21. **kubectl rollout undo**
    Rolls back a deployment to a previous revision.
    Example: `kubectl rollout undo deployment/my-deployment`
22. **kubectl top nodes**
    Shows resource usage for nodes.
    Example: `kubectl top nodes`
23. **kubectl top pods**
    Displays resource usage for pods.
    Example: `kubectl top pods`
24. **kubectl cordon**
    Marks a node as unschedulable.
    Example: `kubectl cordon node-name`
25. **kubectl uncordon**
    Marks a node as schedulable.
    Example: `kubectl uncordon node-name`
26. **kubectl drain**
    Safely evicts all pods from a node.
    Example: `kubectl drain node-name --ignore-daemonsets`
27. **kubectl taint**
    Adds a taint to a node to control pod placement.
    Example: `kubectl taint nodes node-name key=value:NoSchedule`
28. **kubectl get events**
    Lists all events in the cluster.
    Example: `kubectl get events`
29. **kubectl apply -k**
    Applies resources from a kustomization directory.
    Example: `kubectl apply -k ./kustomization-dir/`
30. **kubectl config view**
    Displays the kubeconfig file.
    Example: `kubectl config view`

31. **kubectl config use-context**
    Switches the active context in kubeconfig.
    Example: `kubectl config use-context my-cluster`
32. **kubectl debug**
    Creates a debugging session for a pod.
    Example: `kubectl debug pod-name`
33. **kubectl delete namespace**
    Deletes a namespace and its resources.
    Example: `kubectl delete namespace my-namespace`
34. **kubectl patch**
    Updates a resource using a patch.
    Example: `kubectl patch deployment my-deployment -p '{"spec": {"replicas": 2}}'`
35. **kubectl rollout history**
    Shows the rollout history of a deployment.
    Example: `kubectl rollout history deployment my-deployment`
36. **kubectl autoscale**
    Automatically scales a deployment based on resource usage.
    Example: `kubectl autoscale deployment my-deployment --cpu-percent=50 --min=1 --max=10`
37. **kubectl label**
    Adds or modifies a label on a resource.
    Example: `kubectl label pod pod-name environment=production`
38. **kubectl annotate**
    Adds or modifies an annotation on a resource.
    Example: `kubectl annotate pod pod-name description="My app pod"`
39. **kubectl delete pv**
    Deletes a PersistentVolume (PV).
    Example: `kubectl delete pv my-pv`
40. **kubectl get ingress**
    Lists all Ingress resources in the namespace.
    Example: `kubectl get ingress`
41. **kubectl create configmap**
    Creates a ConfigMap from a file or literal values.
    Example: `kubectl create configmap my-config --from-literal=key1=value1`
42. **kubectl create secret**
    Creates a Secret from a file or literal values.
    Example: `kubectl create secret generic my-secret --from-literal=password=myPassword`
43. **kubectl api-resources**
    Lists all available API resources in the cluster.
    Example: `kubectl api-resources`

44. **kubectl api-versions**
    Lists all API versions supported by the cluster.
    Example: `kubectl api-versions`
45. **kubectl get crds**
    Lists all CustomResourceDefinitions (CRDs).
    Example: `kubectl get crds`

Basic Helm Commands -

Helm is the app store for Kubernetes. It simplifies installing and managing complex applications using pre-packaged "charts" - think of it like apt-get for Kubernetes.

1. **helm help**
   Displays help for the Helm CLI or a specific command.
   Example: `helm help`
2. **helm version**
   Shows the Helm client and server version.
   Example: `helm version`
3. **helm repo add**
   Adds a new chart repository.
   Example: `helm repo add stable https://charts.helm.sh/stable`
4. **helm repo update**
   Updates all Helm chart repositories to the latest version.
   Example: `helm repo update`
5. **helm repo list**
   Lists all the repositories added to Helm.
   Example: `helm repo list`
6. **helm search hub**
   Searches for charts on Helm Hub.
   Example: `helm search hub nginx`
7. **helm search repo**
   Searches for charts in the repositories.
   Example: `helm search repo stable/nginx`
8. **helm show chart**
   Displays information about a chart, including metadata and dependencies.
   Example: `helm show chart stable/nginx`

Installing and Upgrading Charts

9. **helm install**
   Installs a chart into a Kubernetes cluster.
   Example: `helm install my-release stable/nginx`

10. **helm upgrade**
    Upgrades an existing release with a new version of the chart.
    Example: `helm upgrade my-release stable/nginx`
11. **helm upgrade --install**
    Installs a chart if it isn't installed or upgrades it if it exists.
    Example: `helm upgrade --install my-release stable/nginx`
12. **helm uninstall**
    Uninstalls a release.
    Example: `helm uninstall my-release`
13. **helm list**
    Lists all the releases installed on the Kubernetes cluster.
    Example: `helm list`
14. **helm status**
    Displays the status of a release.
    Example: `helm status my-release`

Working with Helm Charts

15. **helm create**
    Creates a new Helm chart in a specified directory.
    Example: `helm create my-chart`
16. **helm lint**
    Lints a chart to check for common errors.
    Example: `helm lint ./my-chart`
17. **helm package**
    Packages a chart into a `.tgz` file.
    Example: `helm package ./my-chart`
18. **helm template**
    Renders the Kubernetes YAML files from a chart without installing it.
    Example: `helm template my-release ./my-chart`
19. **helm dependency update**
    Updates the dependencies in the `Chart.yaml` file.
    Example: `helm dependency update ./my-chart`

Advanced Helm Commands

20. **helm rollback**
    Rolls back a release to a previous version.
    Example: `helm rollback my-release 1`
21. **helm history**
    Displays the history of a release.
    Example: `helm history my-release`

22. **helm get all**
    Gets all information (including values and templates) for a release.
    Example: `helm get all my-release`
23. **helm get values**
    Displays the values used in a release.
    Example: `helm get values my-release`
24. **helm test**
    Runs tests defined in a chart.
    Example: `helm test my-release`

Helm Chart Repositories

25. **helm repo remove**
    Removes a chart repository.
    Example: `helm repo remove stable`
26. **helm repo update**
    Updates the local cache of chart repositories.
    Example: `helm repo update`
27. **helm repo index**
    Creates or updates the index file for a chart repository.
    Example: `helm repo index ./charts`

Helm Values and Customization

28. **helm install --values**
    Installs a chart with custom values.
    Example: `helm install my-release stable/nginx --values values.yaml`
29. **helm upgrade --values**
    Upgrades a release with custom values.
    Example: `helm upgrade my-release stable/nginx --values values.yaml`
30. **helm install --set**
    Installs a chart with a custom value set directly in the command.
    Example: `helm install my-release stable/nginx --set replicaCount=3`
31. **helm upgrade --set**
    Upgrades a release with a custom value set.
    Example: `helm upgrade my-release stable/nginx --set replicaCount=5`

32. **helm uninstall --purge**
    Removes a release and deletes associated resources, including the release history.
    Example: `helm uninstall my-release --purge`

Helm Template and Debugging

33. **helm template --debug**
    Renders Kubernetes manifests and includes debug output.
    Example: `helm template my-release ./my-chart --debug`
34. **helm install --dry-run**
    Simulates the installation process to show what will happen without actually installing.
    Example: `helm install my-release stable/nginx --dry-run`
35. **helm upgrade --dry-run**
    Simulates an upgrade process without actually applying it.
    Example: `helm upgrade my-release stable/nginx --dry-run`

Helm and Kubernetes Integration

36. **helm list --namespace**
    Lists releases in a specific Kubernetes namespace.
    Example: `helm list --namespace kube-system`
37. **helm uninstall --namespace**
    Uninstalls a release from a specific namespace.
    Example: `helm uninstall my-release --namespace kube-system`
38. **helm install --namespace**
    Installs a chart into a specific namespace.
    Example: `helm install my-release stable/nginx --namespace mynamespace`
39. **helm upgrade --namespace**
    Upgrades a release in a specific namespace.
    Example: `helm upgrade my-release stable/nginx --namespace mynamespace`

Helm Chart Development

40. **helm package --sign**
    Packages a chart and signs it using a GPG key.
    Example: `helm package ./my-chart --sign --key my-key-id`
41. **helm create --starter**
    Creates a new Helm chart based on a starter template.

Example: `helm create --starter https://github.com/helm/charts.git`

42. **helm push**
    Pushes a chart to a Helm chart repository.
    Example: `helm push ./my-chart my-repo`

Helm with Kubernetes CLI

43. **helm list -n**
    Lists releases in a specific Kubernetes namespace.
    Example: `helm list -n kube-system`
44. **helm install --kube-context**
    Installs a chart to a Kubernetes cluster defined in a specific kubeconfig context.
    Example: `helm install my-release stable/nginx --kube-context my-cluster`
45. **helm upgrade --kube-context**
    Upgrades a release in a specific Kubernetes context.
    Example: `helm upgrade my-release stable/nginx --kube-context my-cluster`

Helm Chart Dependencies

46. **helm dependency build**
    Builds dependencies for a Helm chart.
    Example: `helm dependency build ./my-chart`
47. **helm dependency list**
    Lists all dependencies for a chart.
    Example: `helm dependency list ./my-chart`

Helm History and Rollbacks

48. **helm rollback --recreate-pods**
    Rolls back to a previous version and recreates pods.
    Example: `helm rollback my-release 2 --recreate-pods`
49. **helm history --max**
    Limits the number of versions shown in the release history.
    Example: `helm history my-release --max 5`

Basic Terraform Commands -

Terraform lets you build cloud infrastructure with code. Instead of clicking buttons in AWS/GCP/Azure consoles, you define servers and services in configuration files.

50**. terraform --help =** Displays general help for Terraform CLI commands.

51. **terraform init =** Initializes the working directory containing Terraform configuration files. It downloads the necessary provider plugins.

52**. terraform validate =** Validates the Terraform configuration files for syntax errors or issues.

53**. terraform plan -** Creates an execution plan, showing what actions Terraform will perform to make the infrastructure match the desired configuration.

54**. terraform apply =** Applies the changes required to reach the desired state of the configuration. It will prompt for approval before making changes.

55**. terraform show =** Displays the Terraform state or a plan in a human-readable format.

56. **terraform output =** Displays the output values defined in the Terraform configuration after an apply.

57**. terraform destroy =** Destroys the infrastructure defined in the Terraform configuration. It prompts for confirmation before destroying resources.

58**. terraform refresh =** Updates the state file with the real infrastructure's current state without applying changes.

59**. terraform taint =** Marks a resource for recreation on the next apply. Useful for forcing a resource to be recreated even if it hasn't been changed.

60**. terraform untaint =** Removes the "tainted" status from a resource.

61**. terraform state =** Manages Terraform state files, such as moving resources between modules or manually

62**. terraform import =** Imports existing infrastructure into Terraform management.

63**. terraform graph =** Generates a graphical representation of Terraform's resources and their relationships.

64**. terraform providers =** Lists the providers available for the current Terraform configuration.

65**. terraform state list =** Lists all resources tracked in the Terraform state file.

66**. terraform backend =** Configures the backend for storing Terraform state remotely (e.g., in S3, Azure Blob Storage, etc.).

67**. terraform state mv =** Moves an item in the state from one location to another.

68**. terraform state rm =** Removes an item from the Terraform state file.

69**. terraform workspace =** Manages Terraform workspaces, which allow for creating separate environments within a single configuration.

70**. terraform workspace new =** Creates a new workspace.

71**. terraform module =** Manages and updates Terraform modules, which are reusable configurations.

72**. terraform init -get-plugins=true =** Ensures that required plugins are fetched and available for modules.

73**. TF_LOG =** Sets the logging level for Terraform debug output (e.g., TRACE, DEBUG, INFO, WARN, ERROR).

74**. TF_LOG_PATH =** Directs Terraform logs to a specified file.

75**. terraform login =** Logs into Terraform Cloud or Terraform Enterprise for managing remote backends and workspaces.

76**. terraform remote =** Manages remote backends and remote state storage for Terraform configurations.

77**. terraform push =** Pushes Terraform modules to a remote module registry.