**Terraform Commands**

**1. terraform init**

* **Purpose**: Initializes the working directory containing Terraform configuration files and sets up the backend and downloads necessary provider plugins.
* **Example**:

**terraform init**

* It’s the first command to run before working with Terraform configurations.

**2. terraform plan**

* **Purpose**: Creates an execution plan, showing what actions Terraform will take to reach the desired state of the infrastructure based on the configuration. Checks for syntax errors
* **Example**:

**terraform plan**

**terraform plan -out=tfplan.out** # Save the plan to a file

* This command allows you to see the changes that will be made to your infrastructure without applying them.

**3. terraform apply**

* **Purpose**: Applies the changes required to reach the desired state of the infrastructure.
* **Example**:

**terraform apply**

**terraform apply tfplan.out** # Apply the plan saved

* It creates or modifies infrastructure according to the plan.

**terraform apply -auto-approve**

* Applies the Terraform plan automatically without requiring manual confirmation.

**4. terraform destroy**

* **Purpose**: Destroys the infrastructure managed by the Terraform configuration. It deletes all the resources defined in the configuration.
* **Example**:

**terraform destroy**

**terraform destroy -auto-approve** # Destroy without asking for confirmation

* Use this command to clean up resources when they are no longer needed.

**5. terraform validate**

* **Purpose**: Validates the configuration files in the directory. It checks the syntax and configuration for potential errors.
* **Example**:

**terraform validate**

* This command ensures that your configuration is syntactically valid before applying it.

**6. terraform fmt**

* **Purpose**: Formats Terraform configuration files to follow a consistent style.
* **Example**:

**terraform fmt**

**terraform fmt -recursive** # Format files recursively in subdirectories

* It’s used to clean up and align your code in a standard way and beautifies the code.

**7. terraform show**

* **Purpose**: Displays the current state or a saved plan in a human-readable format.
* **Example**:

**terraform show**

**terraform show tfplan.out** # Display the saved plan

* This command provides insights into the current state or plan file.

**8. terraform output**

* **Purpose**: Extracts the output values from the Terraform state file. Useful for retrieving values (like IP addresses) after provisioning.
* **Example**:

**terraform output**

**terraform output instance\_ip** # Get the value of a specific output

* This command is used to fetch outputs defined in the configuration.

**9. terraform state**

* **Purpose**: Commands for advanced state management, including viewing, removing, or manipulating resources in the Terraform state.
* **Example**:

**terraform state list** # List all resources in the state

**terraform state show aws\_instance.server\_name** # Show details of a specific resource

**terraform state rm aws\_instance.server\_name** # Remove a resource from the state

* This command is useful when manually interacting with the Terraform state.

**10. terraform refresh**

* **Purpose**: Updates the state file with the real-world infrastructure state. This command fetches the latest data from your cloud provider.
* **Example**:

**terraform refresh**

* It’s used to ensure that the state file matches the current reality of the infrastructure.

**11. terraform taint**

* **Purpose**: Marks a resource for recreation on the next apply. This is useful if you want to force a specific resource to be destroyed and re-created.
* **Example**:

**terraform taint aws\_instance.server\_name** # Mark the instance for recreation

* The marked resource will be destroyed and re-provisioned during the next terraform apply.

**12. terraform untaint**

* **Purpose**: Removes the "tainted" mark from a resource, preventing it from being re-created.
* **Example**:

**terraform untaint aws\_instance.server\_name**

* This command is used when you no longer want a resource to be recreated.

**13. terraform import**

* **Purpose**: Imports an existing infrastructure resource into Terraform's state. It’s useful for managing resources that were created outside of Terraform.
* **Example**:

**terraform import aws\_instance.my\_instance i-1234567890abcdef0**

* This command integrates existing resources into Terraform management.

**14. terraform workspace**

* **Purpose**: Manages multiple workspaces (environments), such as development, staging, and production.
* **Example**:

**terraform workspace list** # List all workspaces

**terraform workspace new dev** # Create a new workspace

**terraform workspace select dev** # Switch to the dev workspace

* Workspaces allow you to manage different versions of infrastructure configurations.

**15. terraform plan -destroy**

* **Purpose**: Generates a destruction plan showing what resources would be destroyed without applying any changes.
* **Example**:

**terraform plan -destroy**

* It’s useful to review which resources will be destroyed before running terraform destroy.

**16.**  **terraform state mv**

* **Purpose**: Moves a resource in the Terraform state file, allowing you to rename or relocate resources.
* **Example**:

**terraform state mv aws\_instance.old aws\_instance.new**

* This is useful when refactoring Terraform configurations or renaming resources.

**17. terraform state pull**

* **Purpose**: Retrieves the current state file from the remote backend.
* **Example**:

**terraform state pull > terraform.tfstate**

* It’s used to download the latest state from the remote backend.

**18. terraform state push**

* **Purpose**: Uploads a local state file to the configured remote backend.
* **Example**:

**terraform state push terraform.tfstate**

* It’s used when you need to push a modified local state to the remote backend.

**19. terraform refresh -lock=false**

* **Purpose**: Updates the Terraform state with the real-world infrastructure, while not locking the state file.
* **Example**:

**terraform refresh -lock=false**

* This is used when you want to avoid locking the state during the refresh process.

**20. terraform force-unlock**

* **Purpose**: Manually unlocks the Terraform state if a previous command caused it to be locked.
* **Example**:

**terraform force-unlock <LOCK\_ID>**

* Use this to unlock a state file when an operation fails and leaves the state locked.

**21. terraform console**

* **Purpose**: Opens an interactive console to evaluate Terraform expressions and inspect resources.
* **Example**:

**terraform console**

* It’s useful for debugging expressions or querying the state in real time.

**22. terraform plan -var-file**

* **Purpose**: Runs terraform plan using a specific variable file. Variable files help separate configuration from code.
* **Example**:

**terraform plan -var-file=prod.tfvars**

* This command allows for variable-driven configurations between environments.

**23. terraform providers**

* **Purpose**: Lists the providers used in the configuration.
* **Example**:  
    
  **terraform providers**
* Shows the cloud providers and other dependencies used in your Terraform configuration.