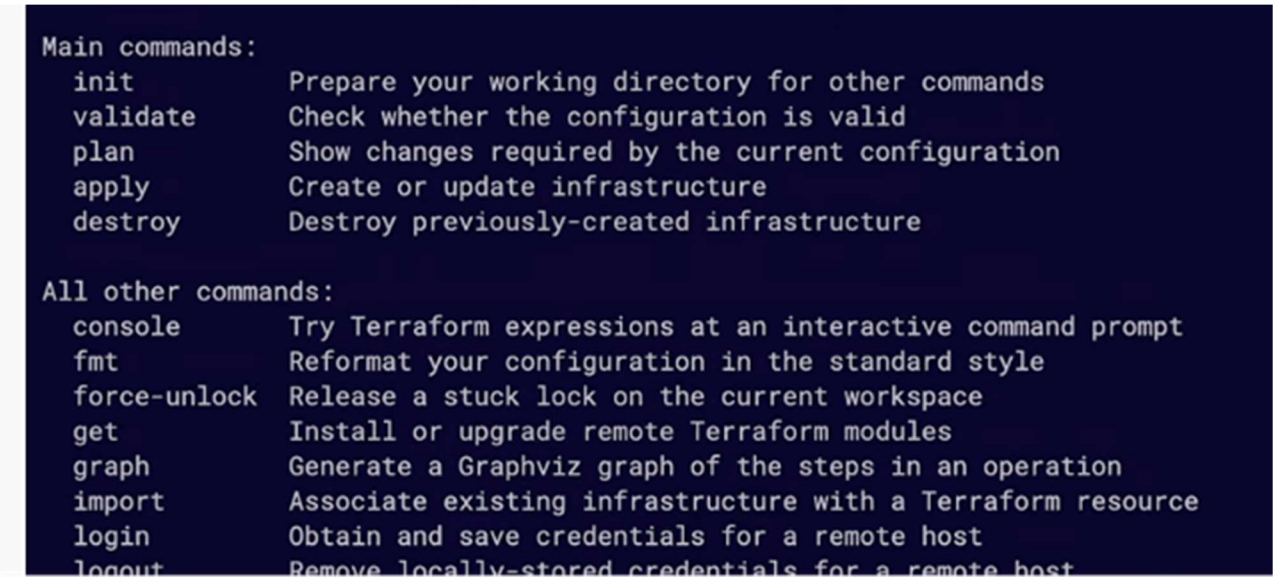
Terraform Commands

There are a couple of commands to check the Terraform's built-in command-line documentation:

- `terraform`
- `terraform -h`
- `terraform --help`

The resulting help page will have the main commands at the top, followed by the less common or more complex commands below.



We can also enter the terraform command and then a subcommand with `-h` or `--help` to pull up a list of commands that are specific to that subcommand.

10 Important Terraform commands:

1. `fmt`

When we finish our Terraform configuration, we can make sure that everything is formatted correctly, run the `fmt` command first.

This command reformats your configuration in the standard style, so it'll make sure that the spacing and everything else is formatted correctly. If it comes back blank, that means the configuration files within your working directory are already correctly formatted. If it does format a file, it will let you know what file it touched.

2. `init`

After we use the `format` command, we have to initialize our working directory to prepare it for what we need.
The `init` command looks at your configuration files and determines which providers and modules it needs to pull down from the registry to allow your configuration to work properly.

3. `validate`

Once we have initialized the directory, it's good to run the`validate` command before you run `plan` or `apply`. Validation will catch syntax errors, version errors, and other issues. One thing to note here is that you can't run `validate` before you run the `init` command. You have to initialize the working directory before you can run the validation.

4. `plan`

Next, it's always a good idea to do a dry run of your plan to see what it's actually going to do. You can even use one of the

subcommands with `terraform plan` to output your plan to apply it later.

5. `apply`

And then of course you have your `apply` command, which is one of the commands you're going to use the most. This is the command that deploys or applies your configuration to a provider.

6. `destroy`

The `destroy` command, obviously, will destroy your infrastructure — or, when used with the `target` flag, individual resources within your infrastructure.

7. `output`

If you've put together a good output variable file, you can use the `output` command to make those defined outputs to display certain information. For example, if you're deploying EC2 instances, you can output tag names, instance names, instance IDs, the IP of the instance, and so on. You can gather some really good information that makes it simple to look up later. And if you're working as a team, people coming behind you can use the `output` command to figure things out and get up to speed.

8. `show`

The `show` command shows the current state of a saved plan, providing good information about the infrastructure you've deployed. For example, if you have an EC2 instance or a VM deployed in your configuration, it'll show you the state that it's in — if it's up and ready or if it's being terminated. It also provides useful information like IP addresses.

9. `state`

Another good way to check your work is to use the `state` command. If you use `state` and then the subcommand `list`, it'll give you a consolidated list of the resources that are being managed by your configuration. If you are moving your Terraform instance, such as from a local instance to a remote backup, you would use the `state mv` command. And just like the `show` command, there's a `state show` command that shows a resource in the state. You can also remove instances from a state by using the `state rm` command.

10. `version`

We will use the `version` command quite a bit to check our Terraform version, especially if we have any version conflicts. Sometimes providers work only with certain versions of Terraform, so if we are defining those versions within our configuration we can use the `version` command.

Terraform CLI tricks

- **`terraform -install-autocomplete`** #Setup tab auto-completion, requires logging back in

Format and Validate Terraform code

- **`terraform fmt`** #format code per HCL canonical standard
- **`terraform validate`** #validate code for syntax
- **`terraform validate -backend=false`** #validate code skip backend validation

Initialize your Terraform working directory

- **`terraform init`** #initialize directory, pull down providers
- **`terraform init -get-plugins=false`** #initialize directory, do not download plugins
- **`terraform init -verify-plugins=false`** #initialize directory, do not verify plugins for Hashicorp signature

Plan, Deploy and Cleanup Infrastructure

1. **`terraform apply --auto-approve`** #apply changes without being prompted to enter "yes"
2. **`terraform destroy --auto-approve`** #destroy/cleanup deployment without being prompted for "yes"
3. **`terraform plan -out plan.out`** #output the deployment plan to plan.out
4. **`terraform apply plan.out`** #use the plan.out plan file to deploy infrastructure
5. **`terraform plan -destroy`** #outputs a destroy plan
6. **`terraform apply -target=aws_instance.my_ec2`** #only apply/deploy changes to the targeted resource
7. **`terraform apply -var my_region_variable=us-east-1`** #pass a variable via command-line while applying a configuration
8. **`terraform apply -lock=true`** #lock the state file so it can't be modified by any other Terraform apply or modification action(possible only where backend allows locking)

9. **`terraform apply refresh=false`** # do not reconcile state file with real-world resources(helpful with large complex deployments for saving deployment time)
10. **`terraform apply --parallelism=5`** #number of simultaneous resource operations
11. **`terraform refresh`** #reconcile the state in Terraform state file with real-world resources
12. **`terraform providers`** #get information about providers used in current configuration

Terraform Workspaces

1. **`terraform workspace new mynewworkspace`** #create a new workspace
2. **`terraform workspace select default`** #change to the selected workspace
3. **`terraform workspace list`** #list out all workspaces

Terraform State Manipulation

1. **`terraform state show aws_instance.my_ec2`** #show details stored in Terraform state for the resource
2. **`terraform state pull > terraform.tfstate`** #download and output terraform state to a file
3. **`terraform state mv aws_iam_role.my_ssm_role module.custom_module`** #move a resource tracked via state to different module
4. **`terraform state replace-provider hashicorp/aws registry.custom.com/aws`** #replace an existing provider with another

5. **`terraform state list`** #list out all the resources tracked via the current state file
6. **`terraform state rm aws_instance.myinstace`** #unmanage a resource, delete it from Terraform state file

Terraform Import And Outputs

1. **`terraform import aws_instance.new_ec2_instance i-abcd1234`** #import EC2 instance with id i-abcd1234 into the Terraform resource named "new_ec2_instance" of type "aws_instance"
2. **`terraform import 'aws_instance.new_ec2_instance[0]' i-abcd1234`** #same as above, imports a real-world resource into an instance of Terraform resource
3. **`terraform output`** #list all outputs as stated in code
4. **`terraform output instance_public_ip`** # list out a specific declared output
5. **`terraform output -json`** #list all outputs in JSON format

Terraform Miscelleneous commands

1. **`terraform version`** #display Terraform binary version, also warns if version is old
2. **`terraform get -update=true`** #download and update modules in the "root" module.

Terraform Console(Test out Terraform interpolations)

1. **`echo 'join(",",["foo","bar"])' | terraform console`** #echo an expression into terraform console and see its expected result as output
2. **`echo '1 + 5' | terraform console`** #Terraform console also has an interactive CLI just enter "terraform console"
3. **`echo "aws_instance.my_ec2.public_ip" | terraform console`** #display the Public IP against the "my_ec2" Terraform resource as seen in the Terraform state file

Terraform Graph(Dependency Graphing)

1. **`terraform graph | dot -Tpng > graph.png`** #produce a PNG diagrams showing relationship and dependencies between Terraform resource in your configuration/code

Terraform Taint/Untaint(mark/unmark resource for recreation -> delete and then recreate)

1. **`terraform taint aws_instance.my_ec2`** #taints resource to be recreated on next apply
2. **`terraform untaint aws_instance.my_ec2`** #Remove taint from a resource
3. **`terraform force-unlock LOCK_ID`** #forcefully unlock a locked state file, LOCK_ID provided when locking the State file beforehand

Terraform Cloud

1. **`terraform login`** #obtain and save API token for Terraform cloud

2. **`terraform logout`** #Log out of Terraform Cloud, defaults to hostname app.terraform.io