Terraform Complete DevOps Reference

========================

1. What is Terraform?

- Open-source Infrastructure as Code (IaC) tool by HashiCorp.
- Automates provisioning and management of infrastructure on cloud providers (AWS, GCP, Azure, etc.) using declarative configuration files.

========================

2. Core Concepts

- Providers: Interface to cloud APIs.
- Resources: Components managed (VM, DB, Network).
- Data Sources: Fetch data for configuration.
- Variables: Parameterize configs.
- Output Values: Display values post deployment.
- State File: Tracks deployed infrastructure.
- Modules: Reusable configuration groups.
- Workspaces: Manage multiple environments.

========================

3. HCL Syntax (Basic)

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}

variable "instance_name" {
  description = "Name of the instance"
  type        = string
}

output "instance_ip" {
  value = aws_instance.example.public_ip
}
```

========================

4. Core Commands

```
terraform init        # Initialize directory
terraform plan        # Preview changes
terraform apply       # Apply changes
terraform destroy     # Remove infra
terraform validate    # Validate syntax
terraform fmt         # Format config files
terraform show        # Show infra state
terraform output      # Display outputs
terraform state list  # List resources in state
```

========================

5. Modules

- Directory with .tf files.
- Called in main config using:

```
module "network" {
  source = "./modules/network"
  vpc_id = var.vpc_id
}
```

========================

6. Remote State

- Store terraform.tfstate remotely (e.g. AWS S3 + DynamoDB lock)

```
backend "s3" {
  bucket = "my-tf-state"
  key    = "state/terraform.tfstate"
  region = "ap-south-1"
}
```

========================

7. Workspaces

- Manage environments (dev, staging, prod)

```
terraform workspace new staging
terraform workspace select staging
terraform workspace list
```

========================

8. Provisioners

- Execute scripts during resource creation.

```
provisioner "local-exec" {
  command = "echo Hello, Terraform"
}
```

========================

9. Best Practices

- Use remote state with locking
- Parameterize with variables
- Organize code with modules
- Use terraform fmt for formatting
- Version control configuration files
- Avoid provisioners where possible

========================

10. Integration in CI/CD

- Use Terraform CLI in pipelines (GitHub Actions, Jenkins, GitLab CI)
- Plan > Approval > Apply pattern

========================

Bonus: State Management Commands

```
terraform state list
terraform state show <resource>
terraform state mv <old> <new>
terraform state rm <resource>
```

Terraform Commands for DevOps

1. terraform init

Syntax: terraform init

Use Case: Initializes a Terraform working directory and downloads the required provider plugins.

Example:

```
terraform init
```

2. terraform plan

Syntax: terraform plan

Use Case: Shows the changes that will be applied without actually applying them.

Example:

```
terraform plan
```

3. terraform apply

Syntax: terraform apply

Use Case: Applies the changes required to reach the desired state of the configuration.

Example:

```
terraform apply
```

4. terraform destroy

Syntax: terraform destroy

Use Case: Destroys all the infrastructure managed by Terraform.

Example:

```
terraform destroy
```

5. terraform validate

Syntax: terraform validate

Use Case: Validates the configuration files for syntax errors and correctness.

Example:

```
terraform validate
```

6. terraform fmt

Syntax: terraform fmt

Use Case: Automatically formats Terraform configuration files to the standard style.

Example:

terraform fmt

7. terraform show

Syntax: terraform show

Use Case: Displays the current state or a saved plan file.

Example:

terraform show

8. terraform state list

Syntax: terraform state list

Use Case: Lists all resources in the current state file.

Example:

terraform state list

9. terraform output

Syntax: terraform output

Use Case: Reads an output variable from a state file.

Example:

terraform output instance_ip

10. terraform taint

Syntax: terraform taint <resource_name>

Use Case: Marks a resource for recreation during the next apply.

Example:

terraform taint aws_instance.my_instance

11. terraform untaint

Syntax: terraform untaint <resource_name>

Use Case: Removes the 'taint' from a resource marked for recreation.

Example:

```
terraform untaint aws_instance.my_instance
```

12. terraform providers

Syntax: terraform providers

Use Case: Displays the providers used in the configuration.

Example:

```
terraform providers
```

13. terraform graph

Syntax: terraform graph

Use Case: Generates a visual graph of Terraform resources and their relationships.

Example:

```
terraform graph | dot -Tpng > graph.png
```