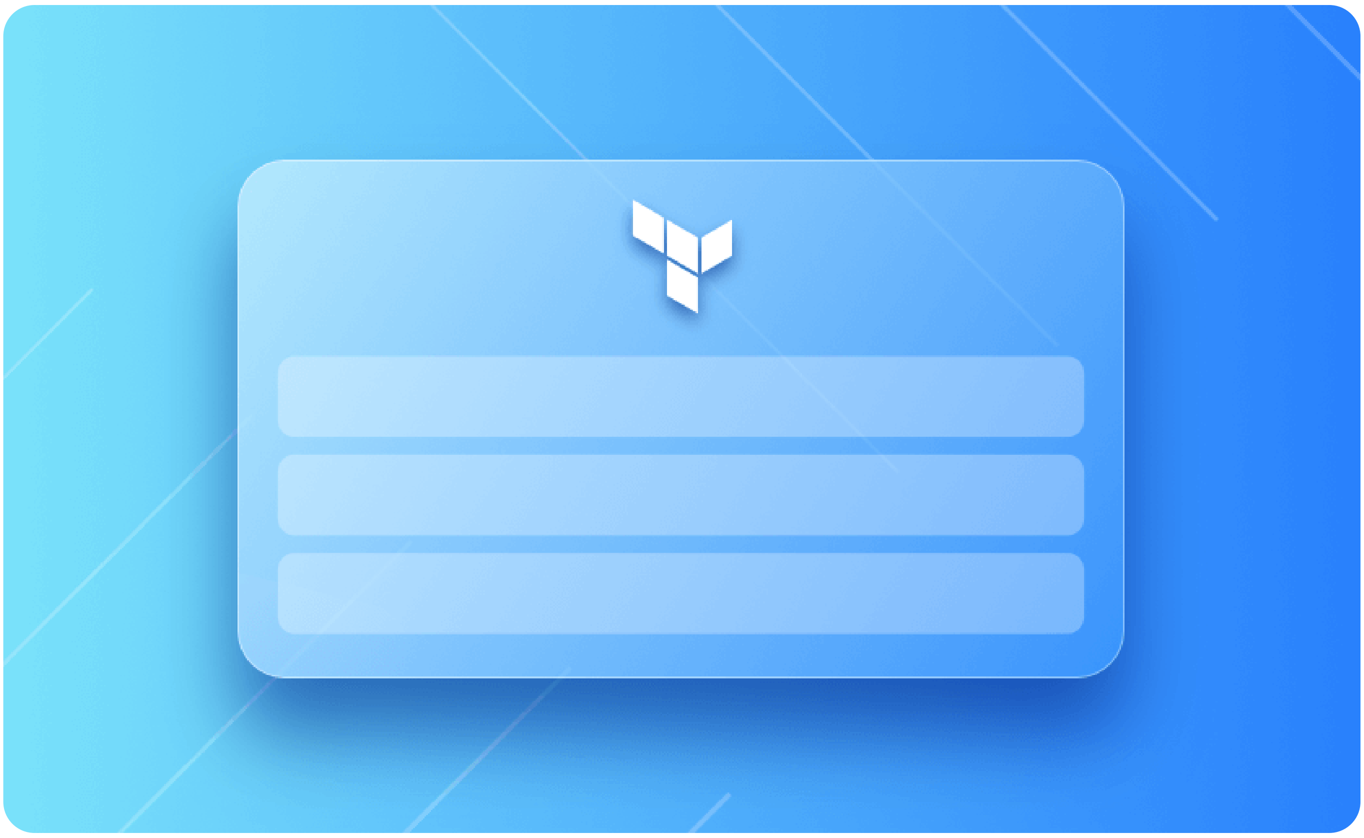
TERRAFORM

Terraform Cheat Sheet – 21 Terraform CLI Commands & Examples

Jack Roper
10 May 2022 · 24 min read



Sometimes you just want to get straight to the commands you need to use with a particular tool, without having to trawl through all the documentation. In this post, I'll highlight the commonly used commands used on the Terraform CLI so you can get straight into the action without the pain! You will also find here a Terraform Cheat Sheet PDF version to download.

This Terraform command reference guide was written using the latest version of Terraform v.1.3.7. New commands and subcommands can be added and depreciated over time with different versions, but should not change too dramatically. You can get the latest version of Terraform here.

Terraform Commands Cheat Sheet:

Get Help

`terraform -help` — Get a list of available commands for execution with descriptions. Can be used with any other subcommand to get more information.

Show Your Terraform Version

`terraform version` — Show the current version of your Terraform and notifies you if there is a newer version available for download.

Format Your Terraform Code

This should be the first command you run after creating your configuration files to ensure your code is formatted using the HCL standards. This makes it easier to follow and aids collaboration.

`terraform fmt` — Format your Terraform configuration files using the HCL language standard.

`terraform fmt --recursive` — Also format files in subdirectories

`terraform fmt --diff` — Display differences between original configuration files and formatting changes.

`terraform fmt --check` — Useful in automation CI/CD pipelines, the check flag can be used to ensure the configuration files are formatted correctly, if not the exit status will be non-zero. If files are formatted correctly, the exit status will be zero.

Initialize Your Directory

`terraform init` — In order to prepare the working directory for use with Terraform, the `terraform init` command performs Backend Initialization, Child Module Installation, and Plugin Installation.

`terraform init -get-plugins=false` — Initialize the working directory, do not download plugins.

`terraform init -input=false` — Initialize the working directory, and disable interactive prompts.

`terraform init -migrate-state` — Reconfigure a backend, and attempt to migrate any existing state.

`terraform init -verify-plugins=false` — Initialize the working directory, do not verify plugins for Hashicorp signature

See our detailed rundown of the `terraform init` command!

Download and Install Modules

Note this is usually not required as this is part of the `terraform init` command.

`terraform get` — Download and installs modules needed for the configuration.

`terraform get -update` — Check the versions of the already installed modules against the available modules and installs the newer versions if available.

Validate Your Terraform Code

`terraform validate` — Validate the configuration files in your directory and does not access any remote state or services. `terraform init` should be run before this command.

`terraform validate -json` — To see easier the number of errors and warnings that you have.

Learn how to validate your configuration locally.

Plan Your Infrastructure

`terraform plan` — Plan will generate an execution plan, showing you what actions will be taken without actually performing the planned actions.

`terraform plan -out=<path>` — Save the plan file to a given path. Can then be passed to the `terraform apply` command.

`terraform plan -destroy` — Create a plan to destroy all objects rather than the usual actions.

Deploy Your Infrastructure

`terraform apply` — Create or update infrastructure depending on the configuration files. By default, a plan will be generated first and will need to be approved before it is applied.

`terraform apply -auto-approve` — Apply changes without having to interactively type 'yes' to the plan. Useful in automation CI/CD pipelines.

`terraform apply <planfilename>` — Provide the file generated using the `terraform plan -out` command. If provided, Terraform will take the actions in the plan without any confirmation prompts.

`terraform apply -lock=false` — Do not hold a state lock during the Terraform apply operation. Use with caution if other engineers might run concurrent commands against the same workspace.

`terraform apply -parallelism=<n>` — Specify the number of operations run in parallel.

`terraform apply -var="environment=dev"` — Pass in a variable value.

`terraform apply -target=”module.appgw.0"` — Apply changes only to the targeted resource.

Destroy Your Infrastructure

`terraform destroy` — Destroy the infrastructure managed by Terraform.

`terraform destroy -target=”module.appgw.0"` — Destroy only the targeted resource.

`terraform destroy --auto-approve` — Destroy the infrastructure without having to interactively type ‘yes’ to the plan. Useful in automation CI/CD pipelines.

`terraform destroy -target="module.appgw.resource[\"key\"]"` — Destroy an instance of a resource created with for_each.

‘Taint’ or ‘Untaint’ Your Resources

Use the `taint` command to mark a resource as not fully functional. It will be deleted and re-created.

`terraform taint vm1.name` — Taint a specified resource instance.

`terraform untaint vm1.name` — Untaint the already tainted resource instance.

Refresh the State File

`terraform refresh` — Modify the state file with updated metadata containing information on the resources being managed in Terraform. Will not modify your infrastructure.

View Your State File

`terraform show` — Show the state file in a human-readable format.

`terraform show <path to statefile>` — If you want to read a specific state file, you can provide the path to it. If no path is provided, the current state file is shown.

Manipulate Your State File

`terraform state` — One of the following subcommands must be used with this command in order to manipulate the state file.

`terraform state list` — Lists out all the resources that are tracked in the current state file.

`terraform state mv` — Move an item in the state, for example, this is useful when you need to tell Terraform that an item has been renamed, e.g. `terraform state mv vm1.oldname vm1.newname`

`terraform state pull > state.tfstate` — Get the current state and outputs it to a local file.

`terraform state push` — Update remote state from the local state file.

`terraform state replace-provider hashicorp/azurerm customproviderregistry/azurerm` — Replace a provider, useful when switching to using a custom provider registry.

`terraform state rm` — Remove the specified instance from the state file. Useful when a resource has been manually deleted outside of Terraform.

Import Existing Infrastructure into Your Terraform State

`terraform import vm1.name -i id123` — Import a VM with id123 into the configuration defined in the configuration files under vm1.name.

See our terraform import tutorial for more details.

Btw. We created a comprehensive pdf version of Terraform Cheatsheet dedicated to those who want to learn and remember the key Terraform commands and have a quick reference guide in pdf form. You can get it below.

Download Terraform Cheat Sheet for Free

Your first name

Your last name

Your business email address

Email me PDF Cheatsheet

Get Provider Information

Manage Your Workspaces

`terraform workspace` — One of the following subcommands must be used with the workspace command. Workspaces can be useful when an engineer wants to test a slightly different version of the code. It is not recommended to use Workspaces to isolate or separate the same infrastructure between different development stages, e.g. Dev / UAT / Production, or different internal teams.

`terraform workspace show` — Show the name of the current workspace.

`terraform workspace list` — List your workspaces.

`terraform workspace select <workspace name>` — Select a specified workspace.

`terraform workspace new <workspace name>` — Create a new workspace with a specified name.

`terraform workspace delete <workspace name>` — Delete a specified workspace.

View Your Outputs

`terraform output` — List all the outputs currently held in your state file. These are displayed by default at the end of a `terraform apply`, this command can be useful if you want to view them independently.

`terraform output -state=<path to state file>` — List the outputs held in the specified state file. **-state** option is ignored when the remote state is used.

`terraform output -json` — List the outputs held in your state file in JSON format to make them machine-readable.

Release a Lock on Your Workspace

`terraform force-unlock <lock_id>` — Remove the lock with the specified lock ID from your workspace. Useful when a lock has become 'stuck', usually after an incomplete Terraform run.

Log In and Out to a Remote Host (Terraform Cloud)

`terraform login` — Grab an API token for Terraform cloud (app.terraform.io) using your browser.

`terraform login <hostname>` — Log in to a specified host.

`terraform logout` — Remove the credentials that are stored locally after logging in, by default for Terraform Cloud (app.terraform.io).

`terraform logout <hostname>` — Remove the credentials that are stored locally after logging in for the specified hostname.

Produce a Dependency Diagram

`terraform graph` — Produce a graph in DOT language showing the dependencies between objects in the state file. This can then be rendered by a program called Graphwiz (amongst others).

`terraform graph -plan=tfplan` — Produce a dependency graph using a specified plan file (generated using `terraform plan -out=tfplan`).

`terraform graph -type=plan` — Specify the type of graph to output, either `plan`, `plan-refresh-only`, `plan-destroy`, or `apply`.

Test Your Expressions

`terraform console` — Allow testing and exploration of expressions on the interactive console using the command line. e.g. 1+2 🙂

Why use Spacelift with Terraform?

Working in a team of Terraform developers can be challenging. Spacelift is built to provide a great CI/CD experience concerning IaC. It can be used for version control of the code, state management, and much more. Spacelift lets you customize the entire infrastructure life-cycle management by providing the ability to run pre and post commands at every stage, which can be very useful in keeping track of things. If you are interested in trying it, create your free trial account.

Key Points

This guide should help you get straight to the command you need when using the Terraform CLI!

If you need more help with Terraform, I encourage you to check the following blog posts: How to Automate Terraform Deployments, and 12 Terraform Best Practices.

Written by

Jack Roper

Jack Roper is a highly experienced IT professional with close to 20 years of experience, focused on cloud and DevOps technologies. He specializes in Terraform, Azure, Azure DevOps, and Kubernetes and holds multiple certifications from Microsoft, Amazon, and Hashicorp. Jack enjoys writing technical articles for well-regarded websites.

spacelift

Product

Documentation

How it works

Spacelift Tutorial

Pricing

Customer Case Studies

Integrations

Security

Product Updates

Company

About Us

Careers

Contact Sales

Partners

Learn

Blog

Spacelift vs Atlantis

Spacelift vs Terraform Cloud

Spacelift for AWS

Get our newsletter Subscribe

Privacy Policy Terms of Service