## Terraform command: init

Usage: terraform [global options] **init** [options]

Initialize a new or existing Terraform working directory by creating initial files, loading any remote state, downloading modules, etc.

This is the first command that should be run for any new or existing Terraform configuration per machine. This sets up all the local data necessary to run Terraform that is typically not committed to version control.

This command is always safe to run multiple times. Though subsequent runs may give errors, this command will never delete your configuration or state. Even so, if you have important information, please back it up prior to running this command, just in case.

## Options:

**-backend=false** Disable backend or Terraform Cloud initialization for this configuration and use what was previously initialized instead.

**aliases: -cloud=false**

**-backend-config=path** Configuration to be merged with what is in the configuration file's 'backend' block. This can be either a path to an HCL file with key/value assignments (same format as terraform.tfvars) or a 'key=value' format, and can be specified multiple times. The backend type must be in the configuration itself.

**-force-copy** Suppress prompts about copying state data when initializating a new state backend. This is equivalent to providing a "yes" to all confirmation prompts.

**-from-module=SOURCE** Copy the contents of the given module into the target directory before initialization.

**-get=false** Disable downloading modules for this configuration.

**-input=false** Disable interactive prompts. Note that some actions may require interactive prompts and will error if input is disabled.

**-lock=false** Don't hold a state lock during backend migration. This is dangerous if others might concurrently run commands against the same workspace.

**-lock-timeout=0s** Duration to retry a state lock.

**-no-color** If specified, output won't contain any color.

**-plugin-dir** Directory containing plugin binaries. This overrides all default search paths for plugins, and prevents the automatic installation of plugins. This flag can be used multiple times.

**-reconfigure** Reconfigure a backend, ignoring any saved configuration.

**-migrate-state** Reconfigure a backend, and attempt to migrate any existing state.

**-upgrade** Install the latest module and provider versions allowed within configured constraints, overriding the default behavior of selecting exactly the version recorded in the dependency lockfile.

**-lockfile=MODE** Set a dependency lockfile mode. Currently only "readonly" is valid.

**-ignore-remote-version** A rare option used for Terraform Cloud and the remote backend only. Set this to ignore checking that the local and remote. Terraform versions use compatible state representations, making an operation proceed even when there is a potential mismatch.

---

## Terraform command: validate

Usage: terraform [global options] validate [options]

Validate the configuration files in a directory, referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc. Validate runs checks that verify whether a configuration is syntactically valid and internally consistent, regardless of any provided variables or existing state. It is thus primarily useful for general verification of reusable modules, including correctness of attribute names and value types.

It is safe to run this command automatically, for example as a post-save check in a text editor or as a test step for a re-usable module in a CI system.

Validation requires an initialized working directory with any referenced plugins and modules installed. To initialize a working directory for validation without accessing any configured remote backend, use:

```
terraform init -backend=false
```

To verify configuration in the context of a particular run (a particular target workspace, input variable values, etc), use the 'terraform plan' command instead, which includes an implied validation check.

## Options:

**-json** Produce output in a machine-readable JSON format, suitable for use in text editor integrations and other automated systems.

Always disables color.

**-no-color** If specified, output won't contain any color.

---

## Terraform command: plan

Usage: terraform [global options] plan [options]

Generates a speculative execution plan, showing what actions Terraform would take to apply the current configuration. This command will not actually perform the planned actions.

You can optionally save the plan to a file, which you can then pass to the "apply" command to perform exactly the actions described in the plan.

Plan Customization Options:

The following options customize how Terraform will produce its plan. You can also use these options when you run "terraform apply" without passing it a saved plan, in order to plan and apply in a single command.

**-destroy** Select the "destroy" planning mode, which creates a plan to destroy all objects currently managed by this Terraform configuration instead of the usual behavior.

**-refresh-only** Select the "refresh only" planning mode, which checks whether remote objects still match the outcome of the most recent Terraform apply but does not propose any actions to undo any changes made outside of Terraform.

**-refresh=false** Skip checking for external changes to remote objects while creating the plan. This can potentially make planning faster, but at the expense of possibly planning against a stale record of the remote system state.

**-replace=resource** Force replacement of a particular resource instance using its resource address. If the plan would've normally produced an update or no-op action for this instance, Terraform will plan to replace it instead. You can use this option multiple times to replace more than one object.

**-target=resource** Limit the planning operation to only the given module, resource, or resource instance and all of its dependencies. You can use this option multiple times to include more than one object. This is for exceptional use only.

**-var 'foo=bar'** Set a value for one of the input variables in the root module of the configuration. Use this option more than once to set more than one variable.

**-var-file=filename** Load variable values from the given file, in addition to the default files terraform.tfvars and *.auto.tfvars. Use this option more than once to include more than one variables file.

**-compact-warnings** If Terraform produces any warnings that are not accompanied by errors, shows them in a more compact form that includes only the summary messages.

**-detailed-exitcode** Return detailed exit codes when the command exits. This will change the meaning of exit codes to:

0 - Succeeded, diff is empty (no changes)

1 - Errored

2 - Succeeded, there is a diff

**-input=true** Ask for input for variables if not directly set.

**-lock=false** Don't hold a state lock during the operation. This is dangerous if others might concurrently run commands against the same workspace.

**-lock-timeout=0s** Duration to retry a state lock.

**-no-color** If specified, output won't contain any color.

**-out=path** Write a plan file to the given path. This can be used as input to the "apply" command.

**-parallelism=n** Limit the number of concurrent operations. Defaults to 10.

**-state=statefile** A legacy option used for the local backend only.

---

## Terraform command: apply

Usage: terraform [global options] apply [options] [PLAN]

Creates or updates infrastructure according to Terraform configuration files in the current directory. By default, Terraform will generate a new plan and present it for your approval before taking any action. You can optionally provide a plan file created by a previous call to "terraform plan", in which case Terraform will take the actions described in that plan without any confirmation prompt.

**Options**:

**-auto-approve** Skip interactive approval of plan before applying.

**-backup=path** Path to backup the existing state file before modifying. Defaults to the "-state-out" path with ".backup" extension. Set to "-" to disable backup.

**-compact-warnings** If Terraform produces any warnings that are not accompanied by errors, show them in a more compact form that includes only the summary messages.

**-lock=false** Don't hold a state lock during the operation. This is dangerous if others might concurrently run commands against the same workspace.

**-lock-timeout=0s** Duration to retry a state lock.

**-input=true** Ask for input for variables if not directly set.

**-no-color** If specified, output won't contain any color.

**-parallelism=n** Limit the number of parallel resource operations.Defaults to 10.

**-state=path** Path to read and save state (unless state-out is specified). Defaults to "terraform.tfstate".

**-state-out=path** Path to write state to that is different than "-state". This can be used to preserve the old state.

If you don't provide a saved plan file then this command will also accept all of the plan-customization options accepted by the terraform plan command.

---

## Terraform command: destroy

Usage: terraform [global options] destroy [options]

Destroy Terraform-managed infrastructure. This command is a convenience alias for:

## terraform apply -destroy

---

## Terraform command: console

Usage: terraform [global options] console [options]

Starts an interactive console for experimenting with Terraform interpolations.

This will open an interactive console that you can use to type interpolations into and inspect their values. This command loads the current state. This lets you explore and test interpolations before using them in future configurations. This command will never modify your state.

## Options:

**-state=path** Legacy option for the local backend only.

**-var 'foo=bar'** Set a variable in the Terraform configuration. This flag can be set multiple times.

**-var-file=foo** Set variables in the Terraform configuration from a file. If "terraform.tfvars" or any ".auto.tfvars" files are present, they will be automatically loaded.

---

## Terraform command: fmt

Usage: terraform [global options] fmt [options] [DIR]

Rewrites all Terraform configuration files to a canonical format. Both configuration files (.tf) and variables files (.tfvars) are updated. JSON files (.tf.json or .tfvars.json) are not modified.

If DIR is not specified then the current working directory will be used. If DIR is "-" then content will be read from STDIN. The given content must be in the Terraform language native syntax; JSON is not supported.

## Options:

**-list=false** Don't list files whose formatting differs (always disabled if using STDIN)

**-write=false** Don't write to source files (always disabled if using STDIN or -check)

**-diff** Display diffs of formatting changes

**-check** Check if the input is formatted. Exit status will be 0 if all input is properly formatted and non-zero otherwise.

**-no-color** If specified, output won't contain any color.

**-recursive** Also process files in subdirectories. By default, only the given directory (or current directory) is processed.

---

## Terraform command: get

Usage: terraform [global options] get [options] PATH

Downloads and installs modules needed for the configuration given by PATH. This recursively downloads all modules needed, such as modules imported by modules imported by the root and so on. If a module is already downloaded, it will not be redownloaded or checked for updates unless the -update flag is specified.

Module installation also happens automatically by default as part of the "terraform init" command, so you should rarely need to run this command separately.

## Options:

**-update** Check already-downloaded modules for available updates and install the newest versions available.

**-no-color** Disable text coloring in the output.

---

## Terraform command: graph

Usage: terraform [global options] graph [options]

Produces a representation of the dependency graph between different objects in the current configuration and state. The graph is presented in the DOT language. The typical program that can read this format is GraphViz, but many web services are also available to read this format.

## Options:

**-plan=tfplan** Render graph using the specified plan file instead of the configuration in the current directory.

**-draw-cycles** Highlight any cycles in the graph with colored edges. This helps when diagnosing cycle errors.

**-type=plan** Type of graph to output. Can be: plan, plan-refresh-only, plan-destroy, or apply. By default Terraform chooses "plan", or "apply" if you also set the -plan=... option.

---

## Terraform command: import

Usage: terraform [global options] import [options] ADDR ID

Import existing infrastructure into your Terraform state. This will find and import the specified resource into your Terraform state, allowing existing infrastructure to come under Terraform management without having to be initially created by Terraform. The ADDR specified is the address to import the resource to. Please see the documentation online for resource addresses. The ID is a resource-specific ID to identify that resource being imported. Please reference the documentation for the resource type you're importing to determine the ID syntax to use. It typically matches directly to the ID that the provider uses.

The current implementation of Terraform import can only import resources into the state. It does not generate configuration. A future version of Terraform will also generate configuration.

Because of this, prior to running terraform import it is necessary to write a resource configuration block for the resource manually, to which the imported object will be attached.

This command will not modify your infrastructure, but it will make network requests to inspect parts of your infrastructure relevant to the resource being imported.

## Options:

**-config=path** Path to a directory of Terraform configuration files to use to configure the provider. Defaults to pwd.If no config files are present, they must be provided via the input prompts or env vars.

**-allow-missing-config** Allow import when no resource configuration block exists.

**-input=false** Disable interactive input prompts.

**-lock=false** Don't hold a state lock during the operation. This is dangerous if others might concurrently run commands against the same workspace.

**-lock-timeout=0s** Duration to retry a state lock.

**-no-color** If specified, output won't contain any color.

**-var 'foo=bar'** Set a variable in the Terraform configuration. This flag can be set multiple times. This is only useful with the "-config" flag.

**-var-file=foo** Set variables in the Terraform configuration from a file. If "terraform.tfvars" or any ".auto.tfvars" files are present, they will be automatically loaded.

**-ignore-remote-version** A rare option used for the remote backend only.

**-state, state-out, and -backup** are legacy options supported for the local backend only.

---

## Terraform command: login

Usage: terraform [global options] login [hostname]

Retrieves an authentication token for the given hostname, if it supports automatic login, and saves it in a credentials file in your home directory.If no hostname is provided, the default hostname is app.terraform.io, to log in to Terraform Cloud.

If not overridden by credentials helper settings in the CLI configuration, the credentials will be written to the following local file:

C:\Users\Anushankar.Konduri\AppData\Roaming\terraform.d\credentials.tfrc.json

---

## Terraform command: logout

Usage: terraform [global options] logout [hostname]

Removes locally-stored credentials for specified hostname.

Note: the API token is only removed from local storage, not destroyed on the remote server, so it will remain valid until manually revoked.

If no hostname is provided, the default hostname is app.terraform.io.

%s

---

## Terraform command: output

Usage: terraform [global options] output [options] [NAME]

Reads an output variable from a Terraform state file and prints the value. With no additional arguments, output will display all the outputs for the root module. If NAME is not specified, all outputs are printed.

## Options:

**-state=path** Path to the state file to read. Defaults to "terraform.tfstate".

**-no-color** If specified, output won't contain any color.

**-json** If specified, machine readable output will be printed in JSON format.

**-raw** For value types that can be automatically converted to a string, will print the raw string directly, rather than a human-oriented representation of the value.

---

## Terraform command: providers

Usage: terraform [global options] providers [dir]

Prints out a tree of modules in the referenced configuration annotated with their provider requirements.

This provides an overview of all of the provider requirements across all referenced modules, as an aid to understanding why particular provider plugins are needed and why particular versions are selected.

Subcommands:

**lock** Write out dependency locks for the configured providers

**mirror** Save local copies of all required provider plugins

**schema** Show schemas for the providers used in the configuration

---

## Terraform command: refresh

Usage: terraform [global options] refresh [options]

Update the state file of your infrastructure with metadata that matches the physical resources they are tracking. This will not modify your infrastructure, but it can modify your state file to update metadata. This metadata might cause new changes to occur when you generate a plan or call apply next.

## Options:

**-compact-warnings** If Terraform produces any warnings that are not accompanied by errors, show them in a more compact form that includes only the summary messages.

**-input=true** Ask for input for variables if not directly set.

**-lock=false** Don't hold a state lock during the operation. This is dangerous if others might concurrently run commands against the same workspace.

**-lock-timeout=0s** Duration to retry a state lock.

**-no-color** If specified, output won't contain any color.

**-parallelism=n** Limit the number of concurrent operations. Defaults to 10.

**-target=resource** Resource to target. Operation will be limited to this resource and its dependencies. This flag can be used multiple times.

**-var 'foo=bar'** Set a variable in the Terraform configuration. This flag can be set multiple times.

**-var-file=foo** Set variables in the Terraform configuration from a file. If "terraform.tfvars" or any ".auto.tfvars" files are present, they will be automatically loaded.

**-state, state-out, and -backup** are legacy options supported for the local backend only.

---

## Terraform command: show

Usage: terraform [global options] show [options] [path]

Reads and outputs a Terraform state or plan file in a human-readable form. If no path is specified, the current state will be shown.

## Options:

**-no-color** If specified, output won't contain any color.

**-json** If specified, output the Terraform plan or state in a machine-readable form.

---

## Terraform command: state

Usage: terraform [global options] state <subcommand> [options] [args]

This command has subcommands for advanced state management. These subcommands can be used to slice and dice the Terraform state. This is sometimes necessary in advanced cases. For your safety, all state management commands that modify the state create a timestamped backup of the state prior to making modifications. The structure and output of the commands is specifically tailored to work well with the common Unix utilities such as grep, awk, etc. We recommend using those tools to perform more advanced state tasks.

Subcommands:

**list** List resources in the state

**mv** Move an item in the state

**pull** Pull current state and output to stdout

**push** Update remote state from a local state file

**replace-provider** Replace provider in the state

**rm** Remove instances from the state

**show** Show a resource in the state

---

## Terraform command: taint

Usage: terraform [global options] taint [options] <address>

Terraform uses the term "tainted" to describe a resource instance which may not be fully functional, either because its creation partially failed or because you've manually marked it as such using this command. This will not modify your infrastructure directly, but subsequent Terraform plans will include actions to destroy the remote object and create a new object to replace it.

You can remove the "taint" state from a resource instance using the "terraform untaint" command.

The address is in the usual resource address syntax, such as:

aws_instance.foo

aws_instance.bar[1]

module.foo.module.bar.aws_instance.baz

Use your shell's quoting or escaping syntax to ensure that the address will reach Terraform correctly, without any special interpretation.

## Options:

**-allow-missing** If specified, the command will succeed (exit code 0) even if the resource is missing.

**-lock=false** Don't hold a state lock during the operation. This is dangerous if others might concurrently run commands against the same workspace.

**-lock-timeout=0s** Duration to retry a state lock.

**-ignore-remote-version** A rare option used for the remote backend only.

**-state, state-out, and -backup** are legacy options supported for the local backend only.

---

## Terraform command: test

Usage: terraform test [options]

This is an experimental command to help with automated integration testing of shared modules. The usage and behavior of this command is likely to change in breaking ways in subsequent releases, as we are currently using this command primarily for research purposes.

In its current experimental form, "test" will look under the current working directory for a subdirectory called "tests", and then within that directory search for one or more subdirectories that contain ".tf" or ".tf.json" files. For any that it finds, it will perform Terraform operations similar to the following sequence of commands in each of those directories:

### terraform validate

### terraform apply

### terraform destroy

The test configurations should not declare any input variables and should at least contain a call to the module being tested, which will always be available at the path ../.. due to the expected filesystem layout. The tests are considered to be successful if all of the above steps succeed.

Test configurations may optionally include uses of the special built-in test provider terraform.io/builtin/test, which allows writing explicit test assertions which must also all pass in order for the test run to be considered successful.

This initial implementation is intended as a minimally-viable product to use for further research and experimentation, and in particular it currently lacks the following capabilities that we expect to consider in later iterations, based on feedback:

- Testing of subsequent updates to existing infrastructure, where currently it only supports initial creation and then destruction.

- Testing top-level modules that are intended to be used for "real" environments, which typically have hard-coded values that don't permit creating a separate "copy" for testing.

- Some sort of support for unit test runs that don't interact with remote systems at all, e.g. for use in checking pull requests from untrusted contributors.

In the meantime, we'd like to hear feedback from module authors who have tried writing some experimental tests for their modules about what sorts of tests you were able to write, what sorts of tests you weren't able to write, and any tests that you were able to write but that were difficult to model in some way.

## Options:

**-compact-warnings** Use a more compact representation for warnings, if this command produces only warnings and no errors.

**-junit-xml=FILE** In addition to the usual output, also write test results to the given file path in JUnit XML format. This format is commonly supported by CI systems, and they typically expect to be given a filename to search for in the test workspace after the test run finishes.

**-no-color** Don't include virtual terminal formatting sequences in the output.

---

## Terraform command: untaint

Usage: terraform [global options] untaint [options] name

Terraform uses the term "tainted" to describe a resource instance which may not be fully functional, either because its creation partially failed or because you've manually marked it as such using the "terraform taint" command. This command removes that state from a resource instance, causing Terraform to see it as fully-functional and not in need of replacement. This will not modify your infrastructure directly. It only avoids Terraform planning to replace a tainted instance in a future operation.

## Options:

**-allow-missing** If specified, the command will succeed (exit code 0) even if the resource is missing.

**-lock=false** Don't hold a state lock during the operation. This is dangerous if others might concurrently run commands against the same workspace.

**-lock-timeout=0s** Duration to retry a state lock.

**-ignore-remote-version** A rare option used for the remote backend only.

**-state, state-out, and -backup** are legacy options supported for the local backend only.

---

## Terraform command: version

Usage: terraform [global options] version [options]

Displays the version of Terraform and all installed plugins

## Options:

**-json** Output the version information as a JSON object.

---

## Terraform command: workspace

Usage: terraform [global options] workspace

new, list, show, select and delete Terraform workspaces.

Subcommands:

**delete** Delete a workspace

**list** List Workspaces

**new** Create a new workspace

**select** Select a workspace

**show** Show the name of the current workspace