HashiCorp Certified: Terraform Associate (2020)

Disclaimer: Below are my personal notes before I sat the exam in Nov 2020. They are in no way comprehensive list or a complete material to help you pass. It only contains bulletpoints which I felt may be important or I had trouble remembering.

1. Semantics:
    - TF loads all the `*.tf` and `*.tf.json` files from the folder in alphabetical order
    - You must not have resources with the same name (e.g. `aws_instance.myec2`)
2. If you have multiple resources already deployed, but you only want to remove one: `terraform destroy -target <resource>.<name>` or comment out with `/* ... */` and re-apply.
3. Fetch the current state of the infrastructure: `terraform refresh`
4. Terraform state file readable format: `terraform show`
5. Desired state is what is explicitly defined in the TF files. If it is not specified, it will not be considered as planned - e.g. If security group is not specified in your TF file, when added in AWS console, TF will NOT remove it because it does not manage it.
6. Always add provider version explicitly, it is a good practice (logical operators are: `~>` , `<` , `>` , `<=` , `=>`)
7. 3rd party provider plugins go to: `%APPDATA%/terraform.d/plugins` on Win, `~/.terraform.d/plugins` on Mac/Linux.
8. If you don't specify attribute of the resource in the `output` - e.g. `public_ip` in `eip.lb.public_ip` , you will get all of them (like `id` , `name` etc.)
9. If you don't define `default` for your variables, you will be asked to provide it in the CLI - e.g.:
    - `variables.tf` :

      ```
      variable <name> {}
      ```
10. You can define variables using env: `export TF_VAR_<variable>="value"`
11. Variable types:
    - `variables.tf` :

      ```
      variable "instance_name" {
          type = number
      }
      ```
    - `terraform.tfvars` - required name!:

      ```
      instance_name = blablabla1
      ```

12. Count index:

    ```
    name  = "loadbalancer.${count.index}"
    count = 5
    ```

    You can also loop throught the list variable, if you don't want index numbers (0, 1, 2, 3...).

13. Conditional expression `condition ? true_value : false_value`:

    ```
    count = var.istest == true ? 1 : 0
    ```

14. Local values:

    ```
    locals {
        common_tags = {
            owner = "DevOps"
            service = "backend"
        }
    }
    ```

    Local can refer to other locals, but reference cycles are not allowed.

15. Functions:

    ```
    terraform console
        max(10,20,30)                       #30
        min(10,20,30)                       #10
        lookup({a="b", c="d"}, "a", "e")    #b
        element(list, index)                #returns element from the list
        file(path)                          #reads content of file
        timestamp()
    ```

16. Data sources read from a specific data source and return the value:

    ```
    data "aws_ami" "app_ami" {
        owners = ["amazon"]
        filter {
            name = "name"
            values = ["amzn2-ami-hvm*"]
        }
    }
    ```

    And to use the value, you would call: `data.aws_ami.app_ami.id`

17. Debugging environment variables:

    - `TF_LOG` with the value of `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`
    - `TF_LOG_PATH` with the path to store the log file

18. Check and fix the indentation: `terraform fmt`
19. Check syntactic validity: `terraform validate`
20. Dynamic blocks inside the resource block:

```
dynamic "ingress" {
    for_each = var.ingress_port
    # iterator = port
    content {
        from_port   = ingress.value # or port.value if iterator is defined
        to_port     = ingress.value # or port.value if iterator is defined
        protocol    = tcp
        cidr_blocks = ["0.0.0.0/0"]
    }
}
```

21. Tainted resource is TF managed resource forced to be destroyed and re-created on the next apply (a.k.a. marked as "must be replaced"): `terraform taint <resource>.<name>`
22. Provisioners:
    - `local-exec` => from where the TF is running
    - `remote-exec` => run on the remote server

```
resource "aws_instance" "web" {
    provisioner "remote-exec" {
        connection {
            type        = ssh
            user        = ec2-user
            private_key = file(path)
            host        = self.public_ip
        }
        inline = ["yum install ..."]
    }
}
```

23. Root & child modules

```
module "myec2" {                        # root module
    source = "../../modules/ec2"    # child module, just a collection of TF
files
}
```

    - When calling modules, you can only overwrite default variables. If the value (e.g. `instance_type`) is hardcoded in the module (e.g. `t2.small`), you cannot overwrite it.
    - Child modules can declare the output values to selectively export certain values to be accessed by the calling module.
24. Registry (https://registry.terraform.io). How to use:

```
module "ec2-instance" {
    source = "terraform-aws-module/ec2-instance/aws"
    version = "2.15.0"  # only works for modules from Terraform registry or TF
cloud
    # insert the 10 required variables here
}
```

And then `terraform init` to initialize.

25. Workspaces - different variables, multiple state files of a single configuration

    ```
    terraform workspace show
    terraform workspace list
    terraform workspace select <name>
    terraform workspace new
    terraform workspace delete
    ```

    For example:

    ```
    instance_type = lookup(var.instance_type, terraform.workspace)

    variable "instance_type" {
        type    = "map"
        default = {
            default = "t2.nano"
            dev     = "t2.micro"
            prd     = "t2.large"
        }
    }
    ```

    - Workspaces crete additional folder: `terraform.tfstate.d/<workspace_name>` , except for the default workspace.

26. `terraform.tfstate` stores everything in clear text, do not store it in git. Store TF files in git, but tfstate in remote backend(s):

    ```
    terraform {
        backend "s3" {
            bucket = "tf-remote-backend-s3"
            key    = "remotedemo.tfstate"
            region = "us-west-2"
        }
    }
    ```

    - And then `terraform init` to initialize.
    - Make sure the backend type supports locking functionality (lock state file for all write operations). Not all of them support it!
    - S3 does not support tfstate locking by default, only when combined with DynamoDB.

27. Terraform state modifications:

```
terraform state list    # shows resources
terraform state mv      # rename without destroy
terraform state pull
terraform state push
terraform state rm      # not removed physically
terraform state show <name>
```

28. To import manually created resources to the state file:

    i. Write TF file that matches the resource.
    ii. Do not run `apply`, otherwise it will create duplicate.
    iii. `terraform import <resource>.<name> <id>`

29. Provider alias to have multiple providers:

```
provider "aws" {
    alias  = mumbai
    region = "ap-south-1"
}
resource "aws_eip" "myeip" {
    vpc      = "true"
    provider = "aws.mumbai"
}
```

30. If you want to hide confidential information (e.g. in the `output`) from showing up in the console: `sensitive = true`

31. Terraform cloud aims to simplify:

    i. Review the TF plan.
    ii. See cost estimation, but only in premium.
    iii. See passed/failed sentinel policies, but only in premium.
    iv. Three choices after that:
        - [Confirm & apply]
        - [Discard run]
        - [Add comment]
    v. Apply running.

    To configure locally:

```
backend.hcl:
    workspaces { name = "demo-repository" }
    hostname     = "app.terraform.io"
    organization = "demo-lmaly"

iam.tf:
    terraform {
        required_version = "~> 0.12.0"
        backend "remote" { }
    }
    resource "aws_iam_user" "lb" {
        name = "remote_user"
        path = "/system/"
    }

terraform login # stored in ~/terraform.d/credentials.tfrc.json
terraform init -backend-config=backend.hcl
```

32. Interpolation syntax deprecation:

```
    v0.11   => "${file("templates/web.tpl")}
    v0.12   =>  "file("templates/web.tpl")
```

33. Help outputs for each command:

- apply (Builds or changes infrastructure)

```
Usage: terraform apply [options] [DIR-OR-PLAN]

  Builds or changes infrastructure according to Terraform configuration
  files in DIR.

  By default, apply scans the current directory for the configuration
  and applies the changes appropriately. However, a path to another
  configuration or an execution plan can be provided. Execution plans can be
  used to only execute a pre-determined set of actions.

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=true             Lock the state file when locking is supported.

  -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.

  -refresh=true          Update state prior to checking for differences. This
                         has no effect if a plan file is given to apply.

  -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.

  -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.
```

- console (Interactive console for Terraform interpolations)

```
Usage: terraform console [options] [DIR]

  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.

  DIR can be set to a directory with a Terraform state to load. By
  default, this will default to the current working directory.

Options:
  -state=path           Path to read state. Defaults to "terraform.tfstate"

  -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.
```

- destroy (Destroy Terraform-managed infrastructure)

```
Usage: terraform destroy [options] [DIR]

  Destroy Terraform-managed infrastructure.

Options:
  -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.

  -auto-approve               Skip interactive approval before destroying.

  -force                      Deprecated: same as auto-approve.

  -lock=true                  Lock the state file when locking is supported.

  -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.

  -refresh=true               Update state prior to checking for differences. This
                              has no effect if a plan file is given to apply.

  -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.

  -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.
```

- env (Workspace management)

```
Usage: terraform 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
```

- fmt (Rewrites config files to canonical format)

```
Usage: terraform 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.
```

- get (Download and install modules for the configuration)

```
Usage: terraform 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.

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

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

- graph (Create a visual graph of Terraform resources)

```
Usage: terraform graph [options] [DIR]

  Outputs the visual execution graph of Terraform resources according to
  configuration files in DIR (or the current directory if omitted).

  The graph is outputted in DOT format. The typical program that can
  read this format is GraphViz, but many web services are also available
  to read this format.

  The -type flag can be used to control the type of graph shown. Terraform
  creates different graphs for different operations. See the options below
  for the list of types supported. The default type is "plan" if a
  configuration is given, and "apply" if a plan file is passed as an
  argument.

Options:
  -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-destroy, apply,
                   validate, input, refresh.

  -module-depth=n  (deprecated) In prior versions of Terraform, specified the
                    depth of modules to show in the output.
```

- import (Import existing infrastructure into Terraform)

```
Usage: terraform 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:
  -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.

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

  -lock=true              Lock the state file when locking is supported.

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

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

  -provider=provider      Deprecated: Override the provider configuration to use
                          when importing the object. By default, Terraform uses
the

                          provider specified in the configuration for the target
                          resource, and that is the best behavior in most cases.

  -state=PATH             Path to the source state file. Defaults to the
configured

                          backend, or "terraform.tfstate"

  -state-out=PATH         Path to the destination state file to write to. If this
                          isn't specified, the source state file will be used.
This

                          can be a new or existing path.

  -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.
```

- init (Initialize a Terraform working directory)

```
Usage: terraform init [options] [DIR]

  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.

  If no arguments are given, the configuration in this working directory
  is initialized.

Options:

  -backend=true        Configure the backend for this configuration.

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

  -force-copy          Suppress prompts about copying state data. 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=true            Download any modules for this configuration.

  -get-plugins=true    Download any missing plugins for this configuration.

  -input=true          Ask for input if necessary. If false, will error if
                       input was required.

  -lock=true           Lock the state file when locking is supported.

  -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 the backend, ignoring any saved
                       configuration.

  -upgrade=false       If installing modules (-get) or plugins (-get-plugins),
                       ignore previously-downloaded objects and install the
                       latest version allowed within configured constraints.

  -verify-plugins=true Verify the authenticity and integrity of automatically
                       downloaded plugins.
```

- login (Obtain and save credentials for a remote host)

```
Usage: terraform 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:
      /home/lmaly/.terraform.d/credentials.tfrc.json
```

- logout (Remove locally-stored credentials for a remote host)

```
Usage: terraform 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
```

- output (Read an output from a state file)

```
Usage: terraform 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
```

- plan (Generate and show an execution plan)

```
Usage: terraform plan [options] [DIR]

  Generates an execution plan for Terraform.

  This execution plan can be reviewed prior to running apply to get a
  sense for what Terraform will do. Optionally, the plan can be saved to
  a Terraform plan file, and apply can take this plan file to execute
  this plan exactly.

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.

  -destroy            If set, a plan will be generated to destroy all resources
                      managed by the given configuration and state.

  -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=true          Lock the state file when locking is supported.

  -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.

  -refresh=true       Update state prior to checking for differences.

  -state=statefile    Path to a Terraform state file to use to look
                      up Terraform-managed resources. By default it will
                      use the state "terraform.tfstate" if it exists.

  -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.
```

- providers (Prints a tree of the providers used in the configuration)

```
Usage: terraform 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:
    schema    Prints the schemas of the providers used in the configuration
```

- refresh (Update local state file against real resources)

```
Usage: terraform refresh [options] [dir]

  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:
  -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.

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

  -lock=true          Lock the state file when locking is supported.

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

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

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

  -state-out=path     Path to write updated state file. By default, the
                      "-state" path will be used.

  -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.
```

- show (Inspect Terraform state or plan)

```
Usage: terraform 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.
```

- taint (Manually mark a resource for recreation)

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

  Manually mark a resource as tainted, forcing a destroy and recreate
  on the next plan/apply.

  This will not modify your infrastructure. This command changes your
  state to mark a resource as tainted so that during the next plan or
  apply that resource will be destroyed and recreated. This command on
  its own will not modify infrastructure. This command can be undone
  using the "terraform untaint" command with the same address.

  The address is in the usual resource address syntax, as shown in
  the output from other commands, such as:
    aws_instance.foo
    aws_instance.bar[1]
    module.foo.module.bar.aws_instance.baz

Options:

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

  -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.

  -lock=true          Lock the state file when locking is supported.

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

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

  -state-out=path     Path to write updated state file. By default, the
                      "-state" path will be used.
```

- untaint (Manually unmark a resource as tainted)

```
Usage: terraform untaint [options] name

  Manually unmark a resource as tainted, restoring it as the primary
  instance in the state.  This reverses either a manual 'terraform taint'
  or the result of provisioners failing on a resource.

  This will not modify your infrastructure. This command changes your
  state to unmark a resource as tainted.  This command can be undone by
  reverting the state backup file that is created, or by running
  'terraform taint' on the resource.

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

  -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.

  -lock=true          Lock the state file when locking is supported.

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

  -module=path        The module path where the resource lives. By
                      default this will be root. Child modules can be specified
                      by names. Ex. "consul" or "consul.vpc" (nested modules).

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

  -state-out=path     Path to write updated state file. By default, the
                      "-state" path will be used.
```

- validate (Validates the Terraform files)

```
Usage: terraform validate [options] [dir]

  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

  If dir is not specified, then the current directory will be used.

  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.
```

- version (Prints the Terraform version)

```
Terraform v0.12.29
```

- workspace (Workspace management)

```
Usage: terraform 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
```

- 0.12upgrade (Rewrites pre-0.12 module source code for v0.12)

```
Usage: terraform 0.12upgrade [module-dir]

  Rewrites the .tf files for a single module that was written for a Terraform
  version prior to v0.12 so that it uses new syntax features from v0.12
  and later.

  Also rewrites constructs that behave differently after v0.12, and flags any
  suspicious constructs that require human review,

  By default, 0.12upgrade rewrites the files in the current working directory.
  However, a path to a different directory can be provided. The command will
  prompt for confirmation interactively unless the -yes option is given.

Options:
  -yes        Skip the initial introduction messages and interactive
              confirmation. This can be used to run this command in
              batch from a script.

  -force      Override the heuristic that attempts to detect if a
              configuration is already written for v0.12 or later.
              Some of the transformations made by this command are
              not idempotent, so re-running against the same module
              may change the meanings expressions in the module.
```

- debug (Debug output management (experimental))

```
Usage: terraform debug <subcommand> [options] [args]
  This command has subcommands for debug output management

Subcommands:
    json2dot    Convert json graph log to dot
```

- force-unlock (Manually unlock the terraform state)

```
Usage: terraform force-unlock LOCK_ID [DIR]

  Manually unlock the state for the defined configuration.

  This will not modify your infrastructure. This command removes the lock on the
  state for the current configuration. The behavior of this lock is dependent
  on the backend being used. Local state files cannot be unlocked by another
  process.

Options:
  -force                 Don't ask for input for unlock confirmation.
```

- push (Obsolete command for Terraform Enterprise legacy (v1))

```
Usage: terraform push [options] [DIR]

  This command was for the legacy version of Terraform Enterprise (v1), which
  has now reached end-of-life. Therefore this command is no longer supported.
```

- state (Advanced state management)

```
Usage: terraform 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
    rm      Remove instances from the state
    show    Show a resource in the state
```

Last Update: Tue Nov 17 10:53:01 UTC 2020