Puppet Declarative config management

START HERE

`puppet parser validate <file.pp>`
Check syntax

`puppet apply --noop <file.pp>`
Dry run locally

`puppet apply <file.pp>`
Apply for real

`puppet agent -t`
One run against server

`puppet resource <type> <name>`
Inspect a resource

`facter <fact>`
Read a system fact

`puppet lookup <key>`
Query Hiera data

`puppet module install <name>`
Get a Forge module

Facts facter on the node → *sent to server* → **Catalog** compiled from manifests + Hiera → *agent fetches* → **Agent** compares to real state → *fixes drift* → **Report** changes and failures

CONCEPTS

Puppet is declarative: you describe the end state and it only changes what differs, so runs are safe to repeat.

Always --noop first: it prints what would change without touching the system.

APPLY AND AGENT

`puppet apply <file.pp>`
Apply a manifest locally

`puppet apply --noop <file.pp>`
Dry run, show changes only

`puppet apply -e '<code>'`
Run a one-line snippet

`puppet apply --test <file.pp>`
Verbose, diffs, detailed exit codes

`puppet apply --detailed-exitcodes <file.pp>`
Exit code reports changes
0 no change, 1 failed, 2 changed, 4 resources failed, 6 changed and failed. Plain apply exits 0 even when it changed things.

`puppet apply --modulepath <dir> <file.pp>`
Use modules from a directory

`puppet apply --tags <tag> <file.pp>`
Only resources with this tag

`puppet apply --debug <file.pp>`
Full debug output

`puppet apply --catalog <file.json>`
Apply a compiled catalog

`puppet agent -t`
One foreground run, verbose

`puppet agent -t --noop`
Preview what the server would change

`puppet agent --onetime --no-daemonize`
Single run, stay in terminal

`puppet agent -t --environment <env>`
Run against another environment

`puppet agent --server <host> -t`
Point at a specific server

`puppet agent --disable "<reason>"`
Pause runs on this node

`puppet agent --enable`
Resume runs

INSPECT AND CHANGE

`puppet resource <type>`
List all resources of a type

`puppet resource <type> <name>`
Show one resource as code

`puppet resource <type> <name> <attr>=<value>`
Change a resource now

`puppet resource service <name> ensure=running`
Start a service

`puppet resource package <name> ensure=present`
Install a package

`puppet resource --types`
List known resource types

`puppet describe <type>`
Docs for a resource type

`puppet describe <type> -s`
Short attribute list

`puppet help <subcommand>`
Help for a subcommand

`puppet --version`
Print Puppet version

FACTS AND CONFIG

`facter`
Print all facts

`facter <fact>`
One fact, e.g. os.family

`facter --json`
All facts as JSON

`facter -p`
Include facts from Puppet plugins

`facter <fact> --debug`
Debug a fact resolving wrong

`puppet facts`
Show facts as the server sees them

`puppet config print`
All settings and values

`puppet config print <setting>`
One setting, e.g. modulepath

`puppet config set <setting> <value>`
Write to puppet.conf

HIERA AND LOOKUP

`puppet lookup <key>`
Look up a data value

`puppet lookup <key> --explain`
Show where the value came from

HIERA AND LOOKUP (cont.)

`puppet lookup <key> --node <node>`
Look up as another node

`puppet lookup <key> --facts <file.yaml>`
Override facts for the lookup

`puppet lookup <key> --merge deep`
Merge hashes and arrays

MODULES

`puppet module install <author-name>`
Install from the Forge

`puppet module install <author-name> --version <ver>`
Pin a version

`puppet module list`
Installed modules

`puppet module list --tree`
Show dependency tree

`puppet module upgrade <author-name>`
Update a module

`puppet module uninstall <author-name>`
Remove a module

`puppet config print modulepath`
Where modules are searched

CHECK AND LINT

`puppet parser validate <file.pp>`
Syntax check only

`puppet-lint <file.pp>`
Style problems

`puppet-lint --fix <file.pp>`
Auto-fix what it can

`puppet-lint --fail-on-warnings <path>`
Non-zero exit for CI

MANIFEST SYNTAX

Resource

```
package { 'nginx':
  ensure ⇒ installed,
}
```

Service and file

```
service { 'nginx':
  ensure ⇒ running,
  enable ⇒ true,
}
file { '/etc/motd':
  ensure  ⇒ file,
  content ⇒ "Managed by
Puppet\n",
}
```

MANIFEST SYNTAX (cont.)

Class

```
class web (
  String $port = '80',
) {
  package { 'nginx': ensure ⇒
installed }
}
include web
```

Ordering

```
package { 'nginx': ensure ⇒
installed }
→ service { 'nginx': ensure ⇒
running }
# or: require ⇒
Package['nginx']
```

Notify on change

```
file { '/etc/nginx/nginx.conf':
  source ⇒ 'puppet:///modules/
web/nginx.conf',
  notify ⇒ Service['nginx'],
}
```

Conditionals

```
if $facts['os']['family'] ==
'Debian' {
  $pkg = 'apache2'
} else {
  $pkg = 'httpd'
}
```

Exec guarded

```
exec { 'reload-units':
  command ⇒ '/bin/systemctl
daemon-reload',
  refreshonly ⇒ true,
}
```

Resource titles must be unique per type. A second package { 'nginx': } anywhere in the catalog is a duplicate declaration error.

Use absolute paths in exec commands, or set path, or Puppet refuses to run them.

RECIPES

Safe local change

```
puppet parser validate site.pp
puppet-lint site.pp
puppet apply --noop site.pp
puppet apply site.pp
```

Preview a node run

```
puppet agent -t --noop
# review, then
puppet agent -t
```

Debug a Hiera value

```
puppet lookup web::port --node
web01 --explain
puppet lookup web::port --facts
test.yaml --explain
```

Test a module quickly

```
puppet module install
puppetlabs-stdlib --target-
dir ./modules
puppet apply --modulepath ./
modules -e 'include stdlib'
```