Resource Types

Note: In these examples, I'm using `should` syntax instead of `expect` syntax because I think `should` syntax is more readable than `expect` syntax and I like it.

Using `expect` syntax is recommended way because adding `should` to every object causes failures when used with BasicObject-subclassed proxy objects.

But the one-liner syntax used with the examples in this page doesn't add `should` to any objects, so this syntax doesn't cause the above problems.That's why I'm using the one-liner `should` syntax.

Please see the document of rspec-expectations if you'd like to know the detail of the issue.

You can use more strict grammar syntax like `be_a_file` instead of `be_file` with all resource types.

---

cgroup

Linux cgroup resource type.

You can test cgroup parameters like this.

```
describe cgroup('group1') do
  its('cpuset.cpus') { should eq 1 }
end
```

---

command

Command resource type.

return_stdout

In order to test a given command returns correct stdout, you should use **return_stdout** matcher.

```
describe command('whoami') do
  it { should return_stdout 'root' }
end
```

You can also use a regular expression.

```
describe command('cat /etc/resolv.conf') do
  it { should return_stdout /8\.8\.8\.8/ }
end
```

return_stderr

In order to test a given command returns correct stderr, you should use **return_stderr** matcher.

```
describe command('ls /foo') do
  it { should return_stderr 'ls: /foo: No such file or directory' }
end
```

You can also use a regular expression.

```
describe command('ls /foo') do
  it { should return_stderr /No such file or directory/ }
end
```

return_exit_status

In order to test a given command returns correct exit status, you should use **return_exit_status** matcher.

```
describe command('ls /tmp') do
  it { should return_exit_status 0 }
end
```

its(:stdout), its(:stderr)

You can get the stdout and stderr, and can use any matchers rspec supports to them.

```
describe command('ls -al /') do
  its(:stdout) { should match /bin/ }
end

describe command('ls /foo') do
  its(:stderr) { should match /No such file or directory/ }
end
```

---

cron

Cron resource type.

have_entry

In order to test cron have a given entry exists, you should use **have_entry** matcher.

```
describe cron do
  it { should have_entry '* * * * * /usr/local/bin/foo' }
end
```

You can test a given user has the cron entry like this.

```
describe cron do
  it { should have_entry('* * * * * /usr/local/bin/foo').with_user('mizzy') }
end
```

---

default_gateway

Default gateway resource type.

In order to test a default gateway is set up correctly, you should use this syntax.

```
describe default_gateway do
  its(:ipaddress) { should eq '192.168.10.1' }
  its(:interface) { should eq 'br0'          }
end
```

---

file

File and directory resource type.

be_file

In order to test a subject exists as a file, you should use **be_file** matcher.

```
describe file('/etc/passwd') do
  it { should be_file }
end
```

be_directory

In order to test a subject exists as a directory, you should use **be_directory** matcher.

```
describe file('/var/log/httpd') do
  it { should be_directory }
end
```

be_socket

In order to test a subject exists as a socket, you should use **be_socket** matcher.

```
describe file('/var/run/unicorn.sock') do
  it { should be_socket }
end
```

contain

Notice: Instead of `contain`, you can use `its(:content)` and any standard rspec matchers. The matcher `contain` will be obsoleted.

```
describe file('/etc/httpd/conf/httpd.conf') do
  its(:content) { should match /ServerName www.example.jp/ }
end
```

In order to test a file contains a given string, you can use **contain** matcher.

```
describe file('/etc/httpd/conf/httpd.conf') do
  it { should contain 'ServerName www.example.jp' }
end
```

You can test a file contains a given string within a given range.

```
describe file('Gemfile') do
  # test 'rspec' exists between "group :test do" and "end".
  it { should contain('rspec').from(/^group :test do/).to(/^end/) }

  # test 'rspec' exists after "group :test do".
  it { should contain('rspec').after(/^group :test do/) }

  # test 'rspec' exists before "end".
  it { should contain('rspec').before(/^end/) }
end
```

be_mode

In order to test a subject is set to given mode, you should use **be_mode** matcher.

```
describe file('/etc/sudoers') do
  it { should be_mode 440 }
end
```

be_owned_by

In order to test a subject is owned by a given user, you should use **be_owned_by** matcher.

```
describe file('/etc/sudoers') do
  it { should be_owned_by 'root' }
end
```

be_grouped_into

In order to test a subject is grouped into a given group, you should use **be_grouped_into** matcher.

```
describe file('/etc/sudoers') do
  it { should be_grouped_into 'wheel' }
end
```

be_linked_to

In order to test a subject is linked to a given file or directory, you should use **be_linked_to** matcher.

```
describe file('/etc/system-release') do
  it { should be_linked_to '/etc/redhat-release' }
end
```

be_readable

In order to test a subject is readable, you should use **be_readable** matcher.

```
describe file('/etc/sudoers') do
  it { should be_readable }
end
```

You can also test a subject is readable by owner, group members, others or a specific user.

```
describe file('/etc/sudoers') do
  it { should be_readable.by('owner') }
  it { should be_readable.by('group') }
  it { should be_readable.by('others') }
  it { should be_readable.by_user('apache') }
end
```

be_writable

In order to test a subject is writable, you should use **be_writable** matcher.

```
describe file('/etc/sudoers') do
  it { should be_writable }
end
```

You can also test a subject is writable by owner, group members, others or a specific user.

```
describe file('/etc/sudoers') do
  it { should be_writable.by('owner') }
  it { should be_writable.by('group') }
  it { should be_writable.by('others') }
  it { should be_writable.by_user('apache') }
end
```

be_executable

In order to test a subject is executable, you should use **be_executable** matcher.

```
describe file('/etc/init.d/httpd') do
  it { should be_executable }
end
```

You can also test a subject is executable by owner, group members, others or a specific user.

```
describe file('/etc/init.d/httpd') do
  it { should be_executable.by('owner') }
  it { should be_executable.by('group') }
  it { should be_executable.by('others') }
  it { should be_executable.by_user('httpd') }
end
```

be_mounted

In order to test a directory is mounted, you should use **be_mounted** matcher.

```
describe file('/') do
  it { should be_mounted }
end
```

You can also test a directory is mounted with correct attributes.

```
describe file('/') do
  it { should be_mounted.with( :type => 'ext4' ) }
end

describe file('/') do
  it { should be_mounted.with( :options => { :rw => true } ) }
end

describe file('/') do
  it do
    should be_mounted.only_with(
      :device  => '/dev/mapper/VolGroup-lv_root',
      :type    => 'ext4',
      :options => {
```

```
        :rw   => true,
        :mode => 620,
      }
    )
  end
end
```

`only_with` needs all attributes of the mounted directory.

match_md5checksum

In order to test a file's md5 checksum matches a given value, you should use **match_md5checksum** matcher.

```
describe file('/etc/services') do
  it { should match_md5checksum '35435ea447c19f0ea5ef971837ab9ced' }
end
```

match_sha256checksum

In order to test a file's sha256 checksum matches a given value, you should use **match_sha256checksum** matcher.

```
describe file('/etc/services') do
  it { should match_sha256checksum 'a861c49e9a76d64d0a756e1c9125ae3aa6b88df3f814a51cecffd3e89cce6210' }
end
```

---

group

Group resource type.

exist

In order to test a group exists, you should use **exist** matcher.

```
describe group('wheel') do
  it { should exist }
end
```

have_gid

In order to test a group have a given gid, you should use **have_gid** matcher.

```
describe group('root') do
  it { should have_gid 0 }
end
```

---

host

Host resource type.

be_resolvable

In order to test a host is resolvable on the target host, you should use **be_resolvable** matcher.

```
describe host('serverspec.org') do
  it { should be_resolvable }
end

describe host('serverspec.org') do
  it { should be_resolvable.by('hosts') }
end

describe host('serverspec.org') do
  it { should be_resolvable.by('dns') }
end
```

be_reachable

In order to test a given host is network reachable, you should use **be_reachable** matcher.

```
describe host('target.example.jp') do
  # ping
  it { should be_reachable }
  # tcp port 22
  it { should be_reachable.with( :port => 22 ) }
  # set protocol explicitly
  it { should be_reachable.with( :port => 22, :proto => 'tcp' ) }
  # udp port 53
  it { should be_reachable.with( :port => 53, :proto => 'udp' ) }
  # timeout setting (default is 5 seconds)
  it { should be_reachable.with( :port => 22, :proto => 'tcp', :timeout => 1 ) }
end
```

its(:ipaddress)

You can get the ipaddress of the host, and can use any matchers rspec supports to them.

```
describe host('example.jp') do
  its(:ipaddress) { should eq '1.2.3.4' }
end
```

```
describe host('example.jp') do
  its(:ipaddress) { should match /1\.2\.3\./ }
end
```

---

interface

Network interface resource type.

In order to test a network interface is set up correctly, you should use this syntax.

```
describe interface('eth0') do
  its(:speed) { should eq 1000 }
end
```

have_ipv4_address

In order to test a interface has a ip address, you should use **have_ipv4_address** matcher.

```
describe interface('eth0') do
  it { should have_ipv4_address("192.168.10.10") }
  it { should have_ipv4_address("192.168.10.10/24") }
end
```

---

ipfilter

Ipfilter resource type.

have_rule

In order to test ipfilter has a given rule, you should use **have_rule** matcher.

```
describe ipfilter do
  it { should have_rule 'pass in quick on lo0 all' }
end
```

---

ipnat

Ipnat resource type.

have_rule

In order to test ipnat has a given rule, you should use **have_rule** matcher.

```
describe ipnat do
  it { should have_rule 'map net1 192.168.0.0/24 -> 0.0.0.0/32' }
end
```

---

iptables

Iptables resource type.

have_rule

In order to test iptables has a given rule, you should use **have_rule** matcher.

```
describe iptables do
  it { should have_rule('-P INPUT ACCEPT') }
end
```

You can give a table name and a chain name like this.

```
describe iptables do
  it { should have_rule('-P INPUT ACCEPT').with_table('mangle').with_chain('INPUT') }
end
```

---

kernel_module

Kernel module resource type.

be_loaded

In order to test a given kernel module is loaded, you should use **be_loaded** matcher.

```
describe kernel_module('virtio_balloon') do
  it { should be_loaded }
end
```

---

linux_kernel_parameter

Linux kernel parameter resource type.

You can test Linux kernel parameters like this.

```
describe 'Linux kernel parameters' do
  context linux_kernel_parameter('net.ipv4.tcp_syncookies') do
    its(:value) { should eq 1 }
  end

  context linux_kernel_parameter('kernel.shmall') do
    its(:value) { should be >= 4294967296 }
  end

  context linux_kernel_parameter('kernel.shmmax') do
    its(:value) { should be <= 68719476736 }
  end

  context linux_kernel_parameter('kernel.osrelease') do
    its(:value) { should eq '2.6.32-131.0.15.el6.x86_64' }
  end

  context linux_kernel_parameter('net.ipv4.tcp_wmem') do
    its(:value) { should match /4096\t16384\t4194304/ }
  end
end
```

LXC

LXC(Linux Container) resource type.

You can test LXC like this.

```
describe lxc('ct01') do
  it { should exist }
  it { should be_running }
end
```

---

mail_alias

Mail alias resource type.

You can test mail aliases like this.

```
describe mail_alias('daemon') do
  it { should be_aliased_to 'root' }
end
```

---

package

Package resource type.

be_installed

In order to test a package is installed, you should use **be_installed** matcher.

```
describe package('httpd') do
  it { should be_installed }
end
```

You can also test a given version of gem is installed.

```
describe package('jekyll') do
  it { should be_installed.by('gem').with_version('0.12.1') }
end
```

---

php_config

PHP config resource type.

You can test PHP config parameters like this.

```
describe 'PHP config parameters' do
  context  php_config('default_mimetype') do
    its(:value) { should eq 'text/html' }
  end

  context php_config('session.cache_expire') do
    its(:value) { should eq 180 }
  end

  context php_config('mbstring.http_output_conv_mimetypes') do
    its(:value) { should match /application/ }
  end
end
```

---

port

Port resource type.

be_listening

In order to test a given port is listening, you should use **be_listening** matcher.

```
describe port(80) do
  it { should be_listening }
end
```

You can also specify `tcp`, `udp`, `tcp6`, or `udp6`.

```
describe port(80) do
  it { should be_listening.with('tcp') }
end

describe port(80) do
  it { should be_listening.with('tcp6') }
end

describe port(53) do
  it { should be_listening.with('udp') }
end

describe port(53) do
  it { should be_listening.with('udp6') }
end
```

---

process

Process resource type.

You can test any process parameter available through the `ps` command like this:

```
describe process("memcached") do
    its(:args) { should match /-c 32000\b/ }
end
```

For the complete list of available parameters, check the manual page for `ps(1)`, section *Standard Format Specifiers*. When several processes match, only the parameters of the first one are be available.

be_running

To check if a given process is running, you should use **be_running** matcher.

```
describe process("memcached") do
  it { should be_running }
end
```

---

routing_table

Routing table resource type.

have_entry

In order to test a routing table has a given entry, you should use **have_entry** matcher.

```
describe routing_table do
  it do
    should have_entry(
      :destination => '192.168.100.0/24',
      :interface   => 'eth1',
      :gateway     => '192.168.10.1',
    )
  end
end
```

---

selinux

SELinux resource type.

be_disabled/be_enforcing/be_permissive

In order to test SELinux is a given mode, you should use **be_disabled, be_enforcing and be_permissive** matchers.

```
# SELinux should be disabled
describe selinux do
  it { should be_disabled }
end

# SELinux should be enforcing
describe selinux do
  it { should be_enforcing }
end

# SELinux should be permissive
describe selinux do
  it { should be_permissive }
end
```

---

service

Service resource type.

be_enabled

In order to test a given service is enabled(automatically start when OS booting up), you should use **be_enabled** matcher.

```
describe service('ntpd') do
  it { should be_enabled }
end
```

You can test a service is enabled with a given run level.(This works only with Red Hat and Debian family currently.)

```
describe service('ntpd') do
  it { should be_enabled.with_level(3) }
end
```

be_running

In order to test a given service/process is running, you should use **be_running** matcher.

```
describe service('ntpd') do
  it { should be_running }
end
```

You can test a given service/process is running under supervisor.

```
describe service('ntpd') do
  it { should be_running.under('supervisor') }
end
```

be_monitored_by

In order to test a service/process is monitored by a given software, you should use **be_monitored_by** matcher.

```
describe service('sshd') do
  it { should be_monitored_by('monit') }
end

describe service('unicorn') do
  it { should be_monitored_by('god') }
end
```

---

user

User resource type.

exist

In order to test a subject exists as a user, you should use **exist** matcher.

```
describe user('root') do
  it { should exist }
end
```

belong_to_group

In order to test a user belongs to a given group, you should use **belong_to_group** matcher.

```
describe user('apache') do
  it { should belong_to_group 'apache' }
end
```

have_uid

In order to test a user have a given uid, you should use **have_uid** matcher.

```
describe user('root') do
  it { should have_uid 0 }
end
```

have_home_directory

In order to test a user have a given home directory, you should use **have_home_directory** matcher.

```
describe user('root') do
  it { should have_home_directory '/root' }
end
```

have_login_shell

In order to test a user have a given login shell, you should use **have_login_shell** matcher.

```
describe user('root') do
  it { should have_login_shell '/bin/bash' }
end
```

have_authorized_key

In order to test a have have a given authorized key, you should use **have_authorized_key** matcher.

```
describe user('root') do
  it { should have_authorized_key 'ssh-rsa ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLM
end
```

---

yumrepo

Yumrepo resource type.

exist

In order to test a given yum repository exists, you should use **exist** matcher.

```
describe yumrepo('epel') do
  it { should exist }
end
```

be_enabled

In order to test a given yum repository is enabled, you should use **be_enabled** matcher.

```
describe yumrepo('epel') do
  it { should be_enabled }
end
```

---

zfs

ZFS resource type.

exist

In order to test a given zfs pool exists, you should use **exist** matcher.

```
describe zfs('rpool') do
  it { should exist }
end
```

have_property

In order to test a zfs pool has given properties, you should use **have_property** matcher.

```
describe zfs('rpool') do
  it { should have_property 'mountpoint' => '/rpool', 'compression' => 'off' }
end
```

1