from AMZ TECHNOLOGIES LLC bookshelf

USER MANUAL

COMPLETE REFERENCE

INCLUDES
+ QUICK START GUIDE
+ END TO END EXAMPLE

kickshell

VERSION 1.0

rev 1a

Table of Contents

QuickStart Guide

1. Verify requirements are met by executing the following command within the local host:

```
for program in perl expect ssh; do type $program > /dev/null; done
```

The above command (bash and ksh) should not print any output. "Requirements" are not met if an error is displayed after executing the command.

To install requirements in **Ubuntu** execute the following command within the local host:

```
sudo apt update && sudo apt install openssh-client perl expect -y
```

To install requirements in **CentOS** execute the following command within the local host:

```
yum install openssh-clients perl-core expect -y
```

2. Download and perform kickshell "Installation". Execute the following commands within the local host:

```
mkdir ~/bin
curl -sL download.kickshell.com --output ~/bin/kickshell
chmod 700 ~/bin ~/bin/kickshell
export PATH=$PATH:~/bin && echo "export PATH=$PATH" >> .profile
```

3. Create a file within the local host named *`run.tsk`* that holds the following contents:

```
desc 'This task will run uptime command and display the result';
task myuptime => sub {
  info run 'uptime';
};

desc 'This task will run uptime command';
task uptime2 => sub {
  run 'uptime';
};
TASKFILE
```

4. Create a file within the local host named *run.ini* that holds similar content to the following (adjust as required):

```
[servers]
your_remoteserver_IP_or_HOSTNAME user=myuser password=mypass
```

5. Execute the following command within the local host:

```
kickshell -e
```

6. When asked, type your profile. If you do not have a profile go to kickshell.com to get more information.

```
provide the registered profile: ***********
```

7. When asked, type your token. If you do not have a token go to kickshell.com to get more information.

```
provide the token: ****
```

8. When asked, type a password of your choice in order to encrypt (and later decrypt) the local file *run.ini* from step 4.

```
write passphrase to encrypt file: ********
```

After successfully encrypting, an additional file *run.grp* will be created, which is the encrypted version of *run.ini* file. The program does not require plain file *run.ini* to execute tasks but it does require encrypted file *run.grp* to run.

9. If step 8 result is "file encrypted successfully" then execute the following command within the local host:

```
kickshell -T
```

You may be asked once every 5 minutes of inactivity for the decryption key (to decrypt temporarily *run.grp*). Type your decryption key (from step 8) when asked:

```
write decrypt passphrase: ********
```

If the command succeeds then the following will be displayed:

```
Tasks
  myuptime------------------>This task will run uptime command and
                             display the result
  uptime2------------------->This task will run uptime command
Server Groups
  servers------------------->your_remoteserver_IP_or_HOSTNAME
```

Two tasks, one group and one server is displayed. Option "T" reads files *run.tsk* and *run.grp* and will display a colored description and summary of the tasks, servers and groups. This will not reflect changes made on *run.ini* until it is re-encrypted to *run.grp* using "e" option as depicted in steps 5 and 8.

10. If step 9 displayed correctly the summary then proceed to type the following command within the local host:

```
kickshell -G servers myuptime
```

This command will execute the task "*myuptime*" defined within *run.tsk* task file on the remote servers contained in group "servers", which in this example is only one server inside one group as stated in *run.ini*, which in fact, what is being really loaded is *run.grp*. In a shared/delegated environment it may be better to delete run.ini once the servers have been set (this may lead to redo and re-encrypt the deleted file if changes are required).

Step 10 may be executed as required, as files *run.grp* and *run.tsk* are all set. If a task has to be edited or added please redo steps 3,9 and 10. If servers, groups, credentials, or other sensitive parameter must be changed then redo steps 4,5,8 and 9.

To execute the task on a host instead of providing the whole group type the following command within the local host:

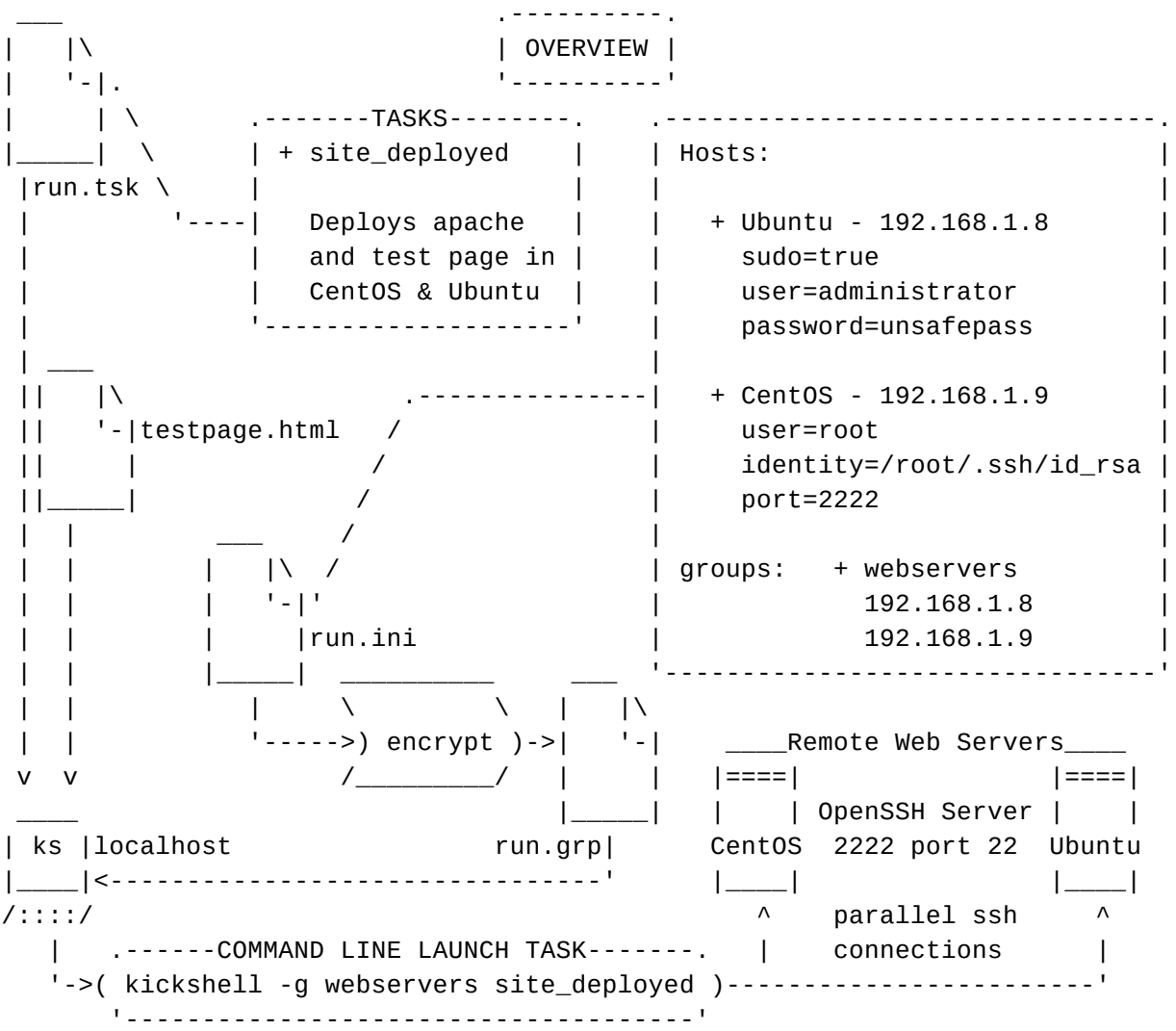
```
kickshell -H your_remoteserver_IP_or_HOSTNAME myuptime
```

End to end example

Unattended deploy Apache Web Server and a test page in CentOS and Ubuntu at once via SSH.

Overview

It is highly recommended to read "QuickStart Guide" before going through this example.



Define task block

Example file *run.tsk*

```
desc 'Deploys apache and test page in CentOS and Ubuntu';          # 01
task site_deployed, sub {                                          # 02
                                                                   # 03
  $indxLocalPath = 'testpage.html';                                # 04
  $indxRemPath = '/var/www/html/testpage.html';                    # 05
                                                                   # 06
  if(OS eq 'CentOS'){                                              # 07
    $owner       = 'apache';                                       # 08
    $packageName = 'httpd';                                        # 09
    $serviceName = 'httpd';                                        # 10
  }elsif(OS eq 'Ubuntu'){                                          # 11
    $owner       = 'www-data';                                     # 12
    $packageName = 'apache2';                                      # 13
    $serviceName = 'apache2';                                      # 14
  }else{                                                           # 15
    die "task only for Ubuntu and CentOS";                         # 16
  }                                                                # 17
                                                                   # 18
  pkg     $packageName, ensure => 'present';                       # 19
                                                                   # 20
  file    $indxRemPath, source => $indxLocalPath,                  # 21
 #                   on_change => sub { $restart = 'true' },       # 22
                         owner => $owner,                          # 23
                          mode => 644;                             # 24
                                                                   # 25
  service $serviceName, 'restart' if $restart;                     # 26
                                                                   # 27
  service $serviceName, ensure => 'started';                       # 28
                                                                   # 29
};                                                                 # 30
                                                                   # 31
TASKFILE                                                           # 32
```

Line by line explanation file *run.tsk*

Lines 1-2

```
desc 'Deploys apache and test page in CentOS and Ubuntu';          # 01
task site_deployed, sub {                                          # 02
```

Line 1 "desc"ribes the task in line 2. Descriptions are displayed when providing "T" option in "Command Line Interface". Line 2 starts the "task" block and sets its "name" as a *bareword* but also starts the "code".

Lines 4-5

```
  $indxLocalPath = 'testpage.html';                                  # 04
  $indxRemPath = '/var/www/html/testpage.html';                      # 05
```

Line 4 defines the variable *indxLocalPath* with the string value ***testpage.html***. Line 5 defines de variable *indxRemPath* with the string value ***/var/www/html/testpage.html***.

Lines 7-17

```
  if(OS eq 'CentOS'){                                                # 07
    $owner       = 'apache';                                         # 08
    $packageName = 'httpd';                                          # 09
    $serviceName = 'httpd';                                          # 10
  }elsif(OS eq 'Ubuntu'){                                            # 11
    $owner       = 'www-data';                                       # 12
    $packageName = 'apache2';                                        # 13
    $serviceName = 'apache2';                                        # 14
  }else{                                                             # 15
    die "task only for Ubuntu and CentOS";                           # 16
  }                                                                  # 17
```

Line 7 executes "OS" "Facts" to verify if the "Operating System" running in the current remote host is *CentOS*. If line 7 is true then *owner*, *packageName* and *serviceName* variables receives the values *apache*, *httpd*, *httpd* respectively in lines 8,9 and 10, as those are the correct values to set in a *CentOS* regarding ownership, package and service for Apache Web Server.

Line 11 executes "OS" "Facts" to verify if the "Operating System" running in the current remote host is *Ubuntu*. If line 11 is true then *owner*, *packageName* and *serviceName* variables receives the values *www-data*, *apache2*, *apache2* respectively in lines 12,13 and 14, as those are the correct values to set in a *Ubuntu* regarding ownership, package and service for Apache Web Server.

Line 15 catches all other cases if the "Operating System" running in the current remote host is not *CentOS* nor *Ubuntu*, and in such case, line 16 halts the execution for the current remote host displaying the error 'task only fo Ubuntu and CentOS'.

Line 19

```
  pkg     $packageName, ensure => 'present';                         # 19
```

Line 19 executes "pkg" command where `$packageName` variable holds one value among these two possible strings: '***httpd***' or '***apache2***'. The effective variable value is determined either on line 9 or line 13, depending on the remote "OS" being traversed by the "task" block at that moment.

The next parameter provided is the literal string '*ensure*', which is a required keyword just before the actual desired state of the package: '***present***'.

Lines 21-24

```
  file    $indxRemPath, source => $indxLocalPath,                 # 21
 #                    on_change => sub { $restart = 'true' },     # 22
                          owner => $owner,                        # 23
                           mode => 644;                           # 24
```

Line 21 executes "file" command where `$indxRemPath` is the variable that holds the string '**/var/www/html/testpage.html**' previously set in line 5. The remote path value has to be an absoulute path.

The next parameter is the literal keyword '*source*' which is required just before the local source path value.

The last parameter in line 21 is the variable `$indxLocalPath` which holds the string '**testpage.html**' previously set in line 4 that corresponds to the path of the local file to be copied to the remote location. The local path may be absolute or relative. A local relative path is relative to the run.tsk file location within the local host.

Line 22 is commented out so it does not set `$restart` variable to '*true*' whenever the source file and remote file are different or the remote file is absent.

Line 23 sets the owner of the file within the remote host, where `$owner` variable holds one value among these two possible strings: '***apache***' or '***www-data***'. The effective variable value is determined either on line 8 or line 12, depending on the remote "OS" being traversed by the "task" block at that moment.

Line 24 sets the remote file permission mode to 644, which corresponds to read and write for owner, read for group and read for everybody else.

Line 26

```
  service $serviceName, 'restart' if $restart;                    # 26
```

Line 26 executes "service" command where `$serviceName` variable holds one value among these two possible strings: '***httpd***' or '***apache2***'. The effective variable value is determined either on line 10 or line 14, depending on the remote "OS" being traversed by the "task" block at that moment.

The next parameter provided is the string '*restart*', which is an action that the Apache Web Service can do in both *CentOS* and *Ubuntu* Operating Systems.

The next word is the reserved '*if*' clause, which verifies if the statement to its right evaluates to a true value. In this example evaluates to '*false*';

Line 28

```
   service $serviceName, ensure => 'started';                    # 28
```

Line 28 executes "service" command where `$serviceName` variable holds one value among these two possible strings: '***httpd***' or '***apache2***'. The effective variable value is determined either on line 10 or line 14, depending on the remote "OS" being traversed by the "task" block at that moment.

The next parameter provided is the literal string '*ensure*', which is a required keyword just before the actual desired state of the service: '***started***'. The service will be started only if it is not running. If the service is started already, no action will be triggered.

Line 30

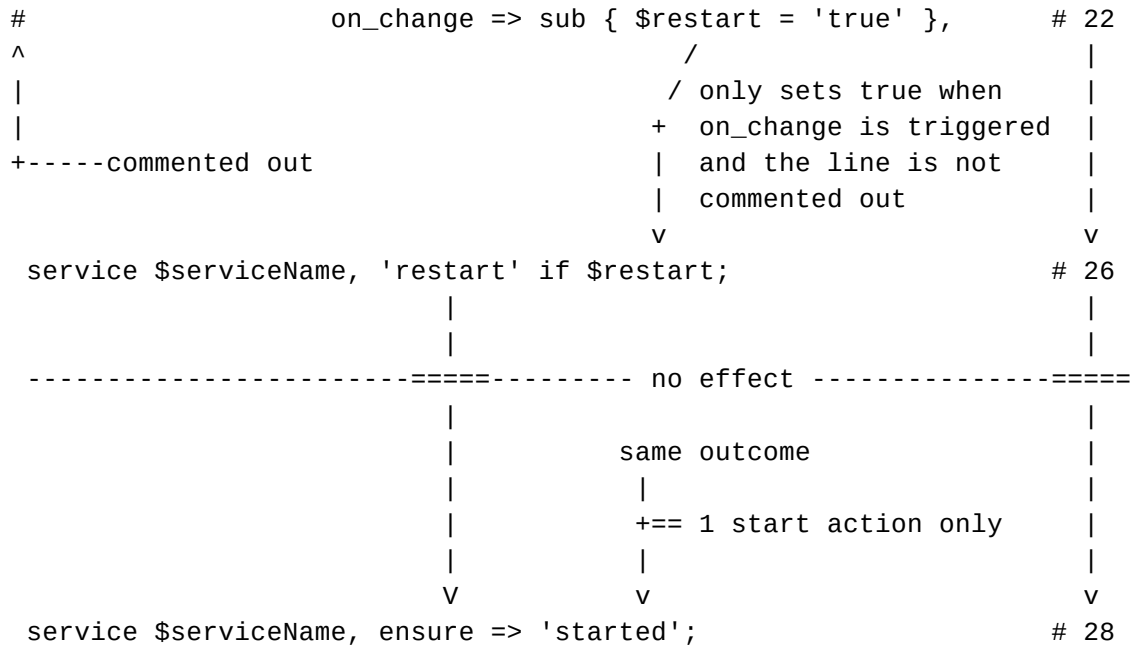
Line 30 contains two characters required in this specific scenario. The closing bracket is required due to the fact that a bracket was opened on line 2 as part of the "code" provided to the "task" command. The semicolon is required at the end of the "task" command as any other statement or command.

Line 32

Line 32 is the last non comment or non blank line and because of that it has to be the exact bareword '*TASKFILE*' to let the program know it has reached the end of the file properly. No semicolon required. Wihtout this, the program will refuse to run.

Important notes and observations on file *run.tsk*

Effect on lines 26 and 28 by line 22



In this specific example, at this specific point in time of execution, where the program reaches line 26, `$restart` variable does not have any value so the '*if*' clause evaluates to '*false*'. This means that the "service" command in line 26 will not be executed due to `$restart` variable evaluating to '*false*'.

A change within a static file such as an html file does not require most Web Servers to restart or even reload its configuration. That is the reason line 22 is commented out. Even if the html file is absent or has changed, the `$restart` variable never gets its '*true*' value, which in turn, will not trigger '*restart*' action within line 26 against the Web Server.

Whether or not `$restart` variable is true or false, it will not change the behavior of line 28. In any scenario, line 28 wil start the service only if it is not started.

In the given case that `$restart` variable is undefined or false, line 26 will not restart or start the Web Service and line 28 will take care. In the given case that `$restart` variable is true, line 26 will restart or start the service which will not change the behavior of line 28 either, due to the fact that line 28 will notice that the service is already started and will not trigger another start action.

In this example, '*on_change*' option from "file" command is not useful as a change in html does not require Web Service restart. However, it may come in handy if the file to be changed or checked for changes is a '*.conf*' file such as *apache.conf* or *httpd.conf*.

Order of the products does matter (in this example)

```
  service $serviceName, 'restart' if $restart;                    # 26

  service $serviceName, ensure => 'started';                      # 28
```

Line 26 has to take place before line 28 otherwise the service may be started and then restarted, causing a total of two service disruptions. Having restart line before started line allows the task to be idempotent.

```
  pkg     $packageName, ensure => 'present';                      # 19
                                                                  # 20
  file    $indxRemPath, source => $indxLocalPath,                 # 21
#                   on_change => sub { $restart = 'true' },       # 22
                        owner => $owner,                          # 23
                         mode => 644;                             # 24
```

Line 19 has to take place before line 21 otherwise the file upload might fail due to the fact that paths might not be present. If the package is installed beforehand then the path will be ready to receive the file.

Define hosts and groups

Example file ***run.ini***

```
[webservers]
192.168.1.8 user=administrator sudo=true password=unsafepass
192.168.1.9 user=root          port=2222 identity=/root/.ssh/id_rsa
```

The text between square brackets is the name of the group and later can be used within the command line with option '**g**' to execute a task on a group of hosts.

The next two lines represent a remote host each. Host parameters must be set within the same line.

In this example file, host 192.168.1.8 has keyword '*sudo*' with an assigned value of '*true*', which is common along *Ubuntu* operating systems, whereas remote host 192.168.1.9 does not, as it is having its user set to root in a *CentOS* operating system.

Remote host 192.168.1.9 has ssh port set to 2222 whereas remote host 192.168.1.8 gets the default ssh port 22 as there is no parameter specifying it.

To authenticate remote hosts, either password or identity parameter is required. Remote host 192.168.1.8 uses password authentication whereas remote host 192.168.1.9 uses RSA public private key authentication.

Static file

Example file *testpage.html*

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <p>This is an example of a simple HTML page with one paragraph.</p>
  </body>
</html>
```

Encrypted file

Generate encrypted file *run.grp*

Once file *run.ini* has been set, file *run.grp* can be generated using the following command:

```
kickshell -e
```

This command will look for the file *run.ini* and encrypt it to *run.grp*. It will ask a passphrase to encrypt the file and it will also ask for a profile and token if not set within *run.ini* file already. If a change is made within file *run.ini* then the encrypt command will have to be run again in order for the *run.grp* file to be updated as well. There is no need to use the encrypt command if the *run.tsk* file has been modified, as it does not contain sensitive data. At this point, if the credentials are required to be shared with a third party, only *run.grp* may be shared as it is encrypted but also due to *run.ini* is not required for runtime.

Verify encrypted file *run.grp*

To verify that *run.grp* is in good order and has the proper and latest contents from the *run.ini* file, the following command may be issued:

```
kickshell -T
```

The command will prompt an error if *run.tsk* or *run.grp* can not be found, otherwise, tasks, hosts and groups will be displayed.

```
Tasks
  site_deployed------------->Deploys apache and test page in CentOS and
                             Ubuntu
Server Groups
  webservers---------------->Ubuntu CentOS
```

Pre-Flight checks

Network communication

The local host needs to be able to communicate with each remote host. The remote hosts may be in different networks and not reachable between them. The communication protocol is TCP/IPv4.

OpenSSH service and port

The remote hosts are required to have a TCP/IPv4 port listening with an OpenSSH service. The local host is required to have OpenSSH client.

Enable passwordless sudo

The software is able to login via password or identity file, however, it is not able to do sudo commands with password. Some Operating Systems, such as *Debian* or *Ubuntu*, have an account with password enabled sudo, different than root to manage the Operating System. It is required that the *NOPASSWD* directive is specified accordingly in the sudoers file (visudo). The following examples will remove the password for sudo calls, which does not remove the authentication method such as password login or identity file login at all.

Full unrestricted passwordless sudo example for administrator user:

```
administrator ALL=(ALL) NOPASSWD: ALL
```

Reduced scope passwordless sudo example for administrator user:

```
administrator ALL=(ALL) NOPASSWD: /usr/bin/systemctl
```

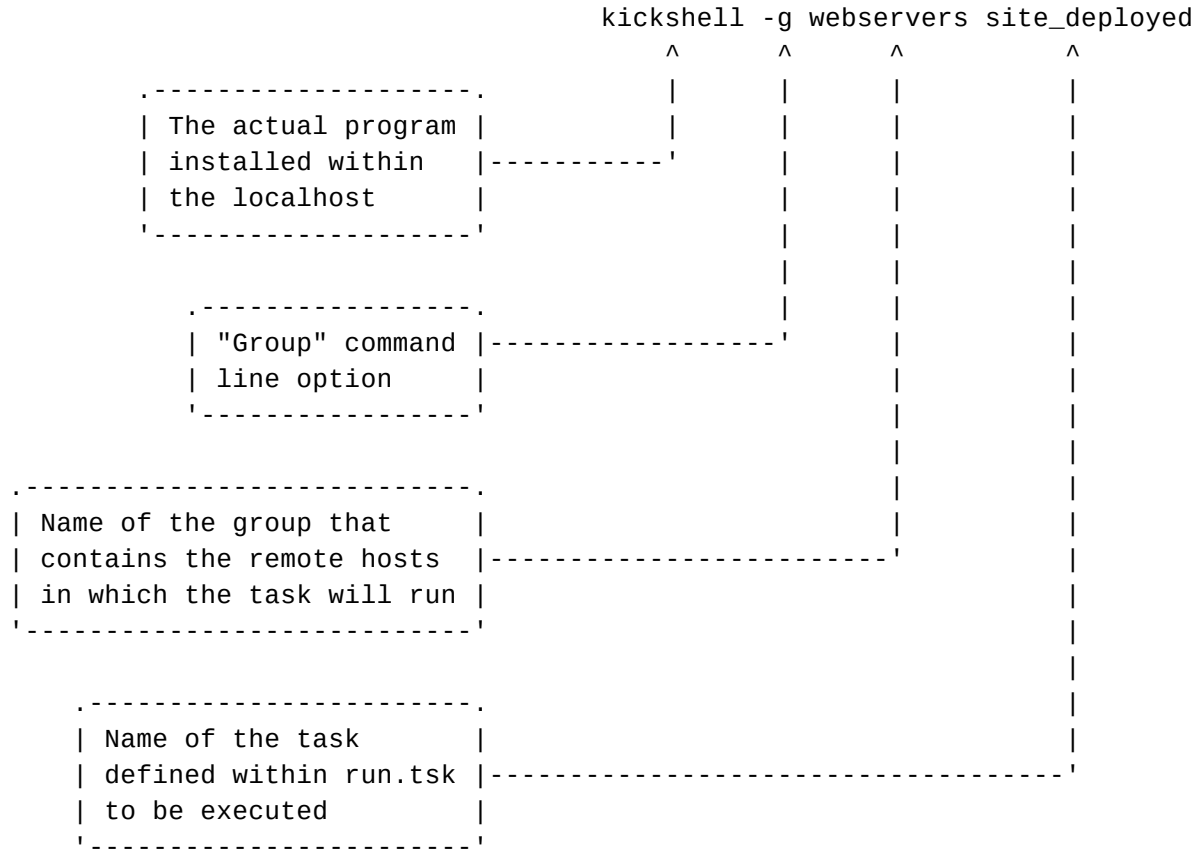
Authorized access

Beware, it is important to double check that each remote host listed within run.ini has the required credentials and proper authorized access, otherwise, the remote host may detect or block unauthorized login attempts from your source local host IP, in addition of any legal action that the other party may take.

Run task

Command Line Launch Task

The following command is the last step to be done in this end to end example. This command can be executed over and over with the same result every time, which is to ensure the site is deployed with the proper package, content and services running, as the task is idempotent.



First attempt: Failure

The following is the output of the first run of the command '*kickshell -g webservers site_deployed*'.

```
Wed May 12 02:45:17 2021 | --START-- | kickshell v1.0
Wed May 12 02:45:17 2021 | Ubuntu | running task 'site_deployed'
Wed May 12 02:45:17 2021 | CentOS | running task 'site_deployed'
Wed May 12 02:45:18 2021 | CentOS | installing httpd
Wed May 12 02:45:38 2021 | Ubuntu | installing apache2
Wed May 12 02:45:47 2021 | CentOS | file 'testpage.html' does not exist
Wed May 12 02:46:12 2021 | Ubuntu | file 'testpage.html' does not exist
Wed May 12 02:46:12 2021 | --FAILED-- | 2/2 failed
```

The program is writing to the console output that 2 out of 2 tasks failed. This information is also written to the system log. Even though the launched task is only one, the total tasks executed are two: one task '*site_deployed*' being executed on Ubuntu remote host, and another task '*site_deployed*' being executed on CentOS remote host.

At Wed May 12 02:45:17 2021, the command was executed, and a few fractions of a second later, both hosts were connected via SSH to the local host. It is common that before a package install in Ubuntu, a package source list update is required, which reflects the delay in starting to install '*httpd*' package in *CentOS* at Wed May 12 02:45:18 2021 and '*apache2*' package in *Ubuntu* at Wed May 12 02:45:38 2021 (20 second delay). Although both remote hosts connected less than a second apart and remain connected in parallel ssh connections to the host, from this point forward, Ubuntu remote host will keep a delay inherent to the package management update until the end of this run.

As there is no package install error message displayed within the console output, it can be assumed that both CentOS and Ubuntu remote hosts had their packages installed successfully. The next run of this task will not install the packages as they will be present from that point forward, unless they are uninstalled before this task runs again.

The next two messages are errors regarding '*testpage.html*'. For this task, as per '*run.tsk*', the source file '*testpage.html*' had to be within the same directory as '*run.tsk*' file due to specifying it as a relative location. The fix is to change it to an absolute location within the task definition or move the file to the same location as the task file.

The last message is the summary of the outcome. For this first attempt, the task failed in both remote hosts at some point along the way, and because of that, the number of failed remote hosts is two out of two remote hosts to execute the task on. The outcome will be '*FAILED*' if at least 1 remote host failed.

Second attempt: Success

The following is the output of the second run of the command '*kickshell -g webservers site_deployed*'.

```
Wed May 12 02:50:55 2021 | --START-- | kickshell v1.0
Wed May 12 02:50:55 2021 | CentOS | running task 'site_deployed'
Wed May 12 02:50:55 2021 | Ubuntu | running task 'site_deployed'
Wed May 12 02:50:58 2021 | CentOS | httpd start
Wed May 12 02:51:05 2021 | -- END -- | success 2/2
```

The outcome for this run is that the task was executed successfully in two remote hosts out of two remote hosts. When the task completes successfully in all remote hosts, the last line printed has the text '*success*'.

In the previous attempt, the task installed the package successfully and failed at the next step. For this second run, the package was already installed, and because of that, the task skipped straight to upload the static file and then service start.

In Ubuntu 20.04, whenever '*apache2*' package is installed, it is automatically started, whereas in CentOS, httpd is not. That is the reason '*apache2* start' does not show up as the service was started just after the package was installed.

Third attempt: Success

```
Wed May 12 02:55:19 2021 | --START-- | kickshell v1.0
Wed May 12 02:55:19 2021 | Ubuntu | running task 'site_deployed'
Wed May 12 02:55:19 2021 | CentOS | running task 'site_deployed'
Wed May 12 02:55:23 2021 | -- END -- | success 2/2
```

At the third attempt, the task completes successfully in both remote hosts within a few seconds. This means that the package is already installed, the html source file is in place and with the proper attributes and the web server is running. From this point forward, the output is going to be the same if these three aspects remain the same.

If the task runs and verifies that the file is different or is absent, or the service is not running or the package is not installed, or any combination of these, the task will set everything in good order.

Requirements

Local

Supported Local Operating Systems

- *Linux*
- *Windows Subsystem for Linux 2 "wsl2"*
- *Unix System V (AIX, HP-UX, Solaris)*

Software / Applications

- *perl*
- *expect*
- *openssh client*

TASKFILE

A '***run.tsk***' file is required within the current working directory. The file may contain "Domain-Specific Language" and Perl programming language. The execution will be halted if the program can not find the file or the last line is not the exact literal string '***TASKFILE***'.

input

- **Files**
  - **ini**

    The file must have ***.ini*** extension. If not specified within the "Command Line Interface", a file with the name ***run.ini*** will be expected to be within the current working directory. Strictly speaking, the plain text ***.ini*** file is not required at runtime as it is only used to generate the encrypted ***.grp*** file.

  - **grp**

    The file must have ***.grp*** extension. If not specified within the "Command Line Interface", a file with the name ***run.grp*** will be expected to be within the current working directory. This file is encrypted and required at runtime.

- **Input file parameter rules**
  - At least one group with one host must be defined.
  - At least user/password or user/identity should be provided per host.
  - A group definition must have its group name between square brackets `[groupnamehere]`.
  - Square brackets are used as 'start' for any group definition and as 'end' of previous group definition.
  - Parameter values must be placed on the right hand side of the equal sign and parameter names must be placed on the left hand side of the equal sign.
    `paramnameonleft=paramvalueonright`
  - One or more spaces are used as a separator between parameters. `param1=value1 param2=value2 paramN=valueN`
  - Lines may be commented out with '`#`' on the left most side of each line.
  - Parameters may be written within the line of a group to apply the parameter to all the hosts defined in that group.
  - Parameters written within the line of a host override those parameters written within the line of its group.
  - Parameters written in latter lines override parameters written in former lines for the same host.
  - Parameter names or values should not contain spaces.
  - A group of groups may be defined using square brackets for the name of the group and writing '@' on the left hand side of the member of the group to be treated as a group.
  - Members with '@' character will not accept parameters.
  - Private or reserved parameters may be set within any group or host line bounded by the same rules as regular parameters.
  - Private parameters may be read within a task via "param".
  - Reserved parameters are recognized within the *input* file and can not be used as private parameters in a task.

- **Reserved parameters**
    - **user**
      username to provide as a credential to login to the remote host
    - **password**
      password to use as a credential to login to the remote host. password and identity parameters are mutually exclusive.
    - **identity**
      identity to use as a credential to login to the remote host. identity and password parameters are mutually exclusive.
    - **port**
      SSH port number that the remote host is listening to. If not specified defaults to 22.
    - **sudo**
      Instructs to execute commands as another user.
    - **sudo_user**
      Sets the user to impersonate within sudo executions.
    - **os**
      Instructs to not identify the remote host "OS" by providing a string to specify the "Operating System".

Remote

Supported Remote Operating Systems

- *CentOS*
- *Ubuntu*
- *SuSE*
- *Redhat*
- *Debian*
- *Fedora*
- *Arch*
- *Oracle*
- *Alpine*
- *Linux*
- *Cygwin*
- *MSYS2*
- *MinGW*
- *AIX*
- *HPUX*
- *SunOS*
- *MacOS*

Software / Applications

- *openssh server*

Installation

local system wide with root or sudo

wget

```
mkdir -p /usr/local/bin
wget -qO /usr/local/bin/kickshell download.kickshell.com
chmod 755 /usr/local/bin/kickshell
```

curl

```
mkdir -p /usr/local/bin
curl -sL download.kickshell.com --output /usr/local/bin/kickshell
chmod 755 /usr/local/bin/kickshell
```

no root in user bin

wget

```
mkdir ~/bin
wget -qO ~/bin/kickshell download.kickshell.com
chmod 700 ~/bin ~/bin/kickshell
export PATH=$PATH:~/bin && echo "export PATH=$PATH" >> .profile
```

curl

```
mkdir ~/bin
curl -sL download.kickshell.com --output ~/bin/kickshell
chmod 700 ~/bin ~/bin/kickshell
export PATH=$PATH:~/bin && echo "export PATH=$PATH" >> .profile
```

Command Line Interface

```
usage: kickshell [options] task [taskParameters]
  -h, --help
       Prints this help
  -d, --disabledomain, --overridedomain [AlternateTokenServer:Port]
       Disables or overrides the default token server run.kickshell.com:80
  -T, -t, --task, --tasks --view --display --show
       Prints available tasks and groups
  -D, --debug
       Enables debug mode
   --nodebug
       Disables debug mode
  -c, --clearcache, --cache
       Clears the stored grp decrypt password within the current shell cache
  -e, --encrypt, --crypt, --export
       Encrypts input file (ini to grp). Defaults to run.ini/run.grp unless
       otherwise specified (see input file option). Asks and updates profile
       and/or token within input file (ini) if not set. Creates (overwrites) a
       filename with grp extension
  -p, --profile
       Asks and updates associated profile within input file (ini). Defaults
       to run.ini unless otherwise specified (see input file option). Implies
       encrypt option
  --token
       Asks and updates the associated token within input file (ini). Defaults
       to run.ini unless otherwise specified (see input file option). Implies
       encrypt option
  -b, --runbefore, --before
       Disables the grp file after the provided date. Hour and minute are set
       to 23:59 if not provided. Updates ini file. Implies encrypt option.
       Defaults to run.ini unless otherwise specified (see input file option)
       Format MM/DD/YYYY-[hh:mm]
  -a, --runafter, --after
       Disables the grp file before the provided date. Hour and minute are set
       to 00:00 if not provided. Updates ini file. Implies encrypt option.
       Defaults to run.ini unless otherwise specified (see input file option)
       Format MM/DD/YYYY-[hh:mm]
  -s, --sum, --checksum
       Adds current tsk checksum into input file (ini) to prevent changes.
       Defaults to run.ini unless otherwise specified (see input file option)
       Implies encrypt option
  -H, --host, --hosts 'host1 host2 host[3..5] host[4,10..19,a..c]app[e,f]'
       Specifies in which hosts the task should run
  -i, --input filename.{ini|grp}
       Sets an alternate input file. Defaults to run.ini/run.grp if not set.
       run.ini file is the "source" file and should be kept secret whereas
       run.grp file can be distributed as it is encrypted
  -G, -g, --group, --groups 'group1 group3 groupN'
       Specifies in which groups of hosts the task should run
  --threads, --parallel, --parallelism numberofconnections
```

```
        Specify the maximum number of parallel SSH connections. Defaults to 35
        if not set. Only one connection per server is made at any given time
   -V  --version
        Displays program version


   Notes: > Do not distribute run.ini file as it may contain plain passwords
          > run.grp is encrypted and can be safely transported by any medium
          > Do not edit run.grp, instead edit run.ini and then use -e switch
          > The encryption key is asked and set every time -e switch is implied
          > There is no way to break the grp file without the encryption key
          > To use a different tsk file change directory and create a new file
          > This program requires tsk and grp files to run
          > This program requires at least one token with available connections
          > The token will be diminished at every run even if the task fails
          > To get a new token or replenish a depleted one please visit
            kickshell.com


# HelloWorld #--run.tsk file example:--
               desc 'This task will run uptime command and display the result';
               task myuptime => sub {
                 info run 'uptime';
               };
               desc 'This task will run uptime command';
               task uptime2 => sub {
                 run 'uptime';
               };
               TASKFILE

             #--.ini file example:--
               [mygrp3]  user=defaultusr password=default sudo=true
               appserv1  sudo=false
               appserv2  user=anotherUser
               dbsrvA    password=Newpassword

             #--kickshell command line examples--
               kickshell -e
               kickshell -T
               kickshell -G mygrp3 myuptime
               kickshell -H 'dbsrvA appserv2' myuptime
               kickshell -H appserv1 uptime2
```

Domain-Specific Language

Set of blocks and reserved keywords to define configuration management, DevOps, application deployment, automation and remote execution via ssh. "Block definitions" must be set in a non recursive way within a "TASKFILE", which in turn, can be used to load supplementary files that may contain additional blocks.

Block definitions

Mandatory block definitions

> task

```
task name[[, boundList], code ] ;

task name[[, group => boundList], code ] ;
```

The *task* block may define "code" to be launched by "name". The block will be executed once for each host selected unless the *task* was called using "runtask". "Commands" and "Facts" can be used only inside a *task* block and will be executed remotely whereas any other code will be executed locally. In order for a task to run, it must have at least 1 host "expand"ed/resolved/selected from the "input" file and provided via "Command Line Interface" or "boundList", otherwise an error will be displayed and the overall execution will be halted. A "desc" block may be defined prior the *task* block to provide a description.

name

The *name* is just a text representation. It can be one of the following:

- **Bareword**

  The bareword must begin with a letter or underscore, in which case it can be arbitrarily long (up to an internal limit of 251 characters) and may contain letters, digits, underscores. The bareword can not be equal to a builtin function name or user defined subroutine name.

- **String**

  Any ASCII character can be used to define a name.

- **SCALAR ref**

  A scalar reference that points to a string.

boundList

A list of hosts (list of strings) or a list of groups (list of strings) may be provided in order to bound to the "task". Every element provided in the *boundList* is subject to "expand". '*group*' key is required before the *boundList* if a list of groups is provided. The *boundList* will be effective only if no host and/or group were selected via "Command Line Interface".

code

A code reference (CodeRef) or an anonymous subroutine of the form `sub{ code }` may be provided. The code will be executed within the current remote host whenever the "task" is launched by its "name".

parameters

Parameters provided to the task are available within the array '`@_`' (at underscore). The first parameter in the array `@_` can be retrieved via `$_[0]`, the second parameter is `$_[1]`, the Nth parameter would be `$_[N-1]`, and the last parameter is `$_[$#_]` due to the fact that `$#_` is the variable that returns the last position within the `@_` array. Parameters can be passed to a task via "runtask" or "Command Line Interface".

> *desc*

```
desc shortDescription ;
```

desc block definition is not a block neither mandatory but its use is highly recommended. It must precede the "task" block definition to describe. *shortDescription* parameter is a string that contains the description of the task to be described within the next code statement.

Optional block definitions

> Always

```
Always code [parameter1, parameter2, parameterN] ;
```

'*Always*' block definition, as its name implies, will be executed whether or not the task execution had a successful outcome. The first parameter is a bare block with *code* in it. The following parameters are optional parameters to pass to the block.

The following example:

```
Always{
  print "this $_[0] always gets $_[1]";
  #do something useful...
} 'message', 'printed';
```

Produces the following output:

```
this message always gets printed
```

> Success

```
Success code [parameter1, parameter2, parameterN] ;
```

'*Success*' block definition will be executed only if the task completes successfully within all remote hosts. The first parameter is a bare block with *code* in it. The following parameters are optional parameters to pass to the block.

The following example adds one line to a local file every time the task is successful:

```
Success{
  print 'appending success log to my local notes...';
  system("echo 'success!!' >> /tmp/localnotes.txt");
};
```

> Failed

```
Failed code [parameter1, parameter2, parameterN] ;
```

'*Failed*' block definition will be executed if the task fails in at least one remote host. The first parameter is a bare block with *code* in it. The following parameters are optional parameters to pass to the block. Example:

```
Failed{
  foreach(@_){
    print "sending failed notification to email $_";
    #code to send email here...
  }
} @emailList;
```

Facts

Networking

> ifname

```
$ifname = ifname
```

Retrieves the ipv4 interface name of the remote host as a string. To gather all ipv4 interface names see "network".

> ip

```
$ipv4 = ip
```

Retrieves the ipv4 of the remote host as a string. To gather all ipv4s see "network".

> subnet

```
$subnet = subnet
```

Retrieves the ipv4 subnet of the remote host as a string. To gather all ipv4 subnets see "network".

> network

```
@network = network [nocache]
```

Retrieves a list (ARRAY) that holds a key value pair (HASH ref) with keys index, ifname, ip and subnet for every ipv4 within the remote host. A re-check can be forced providing a true value for the optional *nocache* parameter.

Startup PID 1

> systemd

```
$systemd = systemd [nocache]
```

Returns true if the remote host is running systemd. A re-check can be forced providing a true value for the optional *nocache* parameter. The startup PID 1 will be identified as needed and stored in cache to prevent further queries.

Operating System

The Operating Systems *Arch*, *Alpine*, *SuSE*, *Redhat*, *Debian*, *CentOS*, *Ubuntu*, *Fedora* and *Oracle* are fully supported. The Operating Systems *MSYS2*, *MinGW* and *Cygwin* do not support "service" command. The Operating Systems *MinGW* and *Cygwin* do not support "pkg" command. Other Operating Systems may or not be compatible with some commands. A workaround may be achieved by setting the cache as a similar OS. The remote host "OS" will be identified as needed and stored in cache to prevent further queries.

> OS cache

The following fact cache setters are available to manually specify the remote "Operating System".

- Arch
- Alpine
- SuSE
- Redhat
- Debian
- MinGW
- MSYS2
- Cygwin
- CentOS
- Ubuntu
- Fedora
- Oracle

> OS

```
$OS = OS [nocache]
```

Identifies and returns the remote host "Operating System" as a string. Returns an empty string if the Operating System could not be determined. A re-check can be forced providing a true value for the optional *nocache* parameter. Called internally from "pkg", "service" and "OSversion".

Possible OS string return values

See "Supported Remote Operating Systems".

> OSversion

```
$OSversion = OSversion [nocache]
```

Identifies the remote host "OS" and returns the remote host Operating System version as a string. Returns an empty string if the "Operating System" or Operating System version could not be determined. A re-check can be forced providing a true value for the optional *nocache* parameter. *OSversion* functionality is limited to *CentOS*, *Redhat*, *Ubuntu*, *SuSE*, *Alpine*, AIX, HPUX, Solaris and MacOS Operating Systems only.

Helpers

Information

> $HOST

`$HOST` variable holds the current remote host as named within the "input" file. It contains a string with the name of the host being worked on.

> param

%param hash holds the private variables for the current remote *$HOST*. `$param{var}` may be used to access a specific private variable where '*var*' is the name of the parameter. The return value is the string provided within the "input" file.

> info

`info` *`stringToDisplay`*

Prints green colored output with the local time, the remote host and the *stringToDisplay*. Data is sent to syslog as '*info*'.

> warn

`warn` *`stringToDisplay`*

Prints yellow colored output with the local time, the remote host and the *stringToDisplay*. It will also display the line number in which it was called unless a new line character is put at the end of the *stringToDisplay*. Data is sent to syslog as '*warning*'.

> die

`die` *`stringToDisplay`*

Prints red colored output with the local time, the remote host and the *stringToDisplay*. It will also display the line number in which it was called unless a new line character is put at the end of the *stringToDisplay*. Data is sent to syslog as '*error*'. **The execution will be halted.**

Customization

> template

```
$text = template pathToLocalTemplate, %variables
```

Produces text from a local template file that may contain variables to be rendered.

The variable substitution can be achieved by writing one or more times within the template the following example snippet:

```
<%= $variableName %>
```

Where `<%=` acts as the initial delimiter, `%>` acts as the ending delimiter, and **`$variableName`** is the variable provided within the **`%variables`** parameter list (hash) in which the key acts as the variable name as identified in the template, and the value is the literal string to be substituted within the template.

```
%variables
```

key=>***value*** can be seen as **variableName**=>***valueToBeSubstituted***

Example

Given the following local file contained in '*mylocaltemplates*' directory in localhost:

```
[www]
user = <%= $user %>
group = <%= $group %>
listen = 127.0.0.1:<%= $port %>
pm = dynamic
pm.max_children = 5
pm.start_servers = 2
pm.min_spare_servers = 1
pm.max_spare_servers = 3
security.limit_extensions =
```

And having the following code within a run.tsk file:

```
file '/etc/php7/php-fpm.d/www.conf',
  content   => template(  'mylocaltemplates/www.conf',
                            user  => 'nginx',
                            group => 'nginx',
                            port  => 9000),
  owner     => 'nginx',
  group     => 'nginx',
  mode      => 600,
  on_change => sub { $restart = 'true' };
```

The following resulting file would be uploaded to the remote host into the path */etc/php7/php-fpm.d/www.conf* if absent or different:

```
[www]
user = nginx
group = nginx
listen = 127.0.0.1:9000
pm = dynamic
pm.max_children = 5
pm.start_servers = 2
pm.min_spare_servers = 1
pm.max_spare_servers = 3
security.limit_extensions =
```

Transportation

> fetch

```
$localPath = fetch $remoteFile[, $suffix]
```

Retrieves *`$remoteFile`* from each remote host and returns *`$localPath`* to the fetched file. The execution will be halted and the error will be displayed if the fetch fails. An optional *`$suffix`* can be appended to the *`$localPath`*.

Delegation

> runtask

```
$returnValue = runtask taskName[, taskParameters][, on=>target]

@returnValues = runtask taskName[, taskParameters][, on=>target]
```

Executes and waits the task specified in *taskName*. The task may or not halt the execution.

taskName

The "name" of the "task" to be executed.

taskParameters

"parameters" of any type may be provided to be passed to the task.

target

Enables serial delegation. If no target is specified it defaults to the current `$HOST`. '*on*' key is required before defining the *target*. The value provided is subject to "expand" and it may be one of the following:

- **host**

  Host mode expects a string with the host to delegate the task to.

- **group**

  This mode expects an ArrayRef with hosts (strings) and/or groups (ArrayRefs) to delegate the task to. It can be a variable pointing to an ArrayRef, an anonymous ArrayRef or a named group using the 'group' statement (which retrieves the ArrayRef from the named group).

- **predefined**

  Predefined mode expects the literal string 'predefined' which delegates the task to the hosts or groups previously set within the "boundList" task declaration.

If the target represents one host then the `$returnValue` or `@returnValues` is/are what the task itself returns as the calling context is passed on to the task. If the target represents more than one host then the `@returnValues` will be a list of three ArrayRefs (result, exit, name) if called in list context or will be an ArrayRef `$returnValue` that holds the same three ArrayRefs in it if called in scalar context.

- **result**

  This ArrayRef holds the *output* of each host. Element in position zero corresponds to the *output* gathered from the first host, element in position one corresponds to the *output* gathered from the second host, etc.

- **exit**

  This ArrayRef holds the *exitcode* of each host. Element in position zero corresponds to the *exitcode* gathered from the first host, element in position one corresponds to the *exitcode* gathered from the second host, etc.

- **name**

  This ArrayRef holds the *identifier* of each host. Element in position zero corresponds to the *identifier* gathered from the first host, element in position one corresponds to the *identifier* gathered from the second host, etc.

Impersonation

> sudo

```
$sudoStatus = sudo [codeBlock [sudoUser]]

$sudoStatus = sudo [\&subName [,sudoUser]]

$sudoStatus = sudo [sub {run 'echo sudo_test'} [,sudoUser]]
```

Executes commands as another user. The current `$sudoStatus` will be toggled between enabled or disabled if no parameters are provided. If *codeBlock* or a subroutine is provided then sudo will be enabled temporarily for the execution of the code specified within the *codeBlock* or subroutine and restored to its value prior execution. Returns the current `$sudoStatus` where 0 is sudo disabled.

codeBlock

Code between `{` and `}` to be executed as another user.

```
{ run 'echo sudo_test' }
```

sudoUser

If *codeBlock* or a subroutine is provided then *sudoUser* may be specified and used temporarily otherwise the previously set value will be used.

Aggregation

> group

```
$groupRef = group name[, memberList]
```

Returns an array reference (ArrayRef) listing the members of the group or 'undefined' if parameters are not provided or the group does not exist. The first parameter is the "name" of the group. The next optional parameters are used to specify the members of the group.

Commands

--- Stateless ---

> run

```
$output = run $shellcode

( $output, $exitcode ) = run $shellcode
```

Executes on the remote host the code provided as a string. Sets the exit code of the last command executed in the $? variable. In scalar context returns the output of the execution as a string. In list context returns the output of the execution and the exit code of the last command executed as a two item list. The provided parameter will be displayed if debug mode is enabled.

> runok

```
$runok = runok $shellcode
```

Executes on the remote host the code provided as a string. Sets the exit code of the last command executed in the $? variable. Returns true when the exit code of the last command executed is equal to zero. The provided parameter will be displayed if debug mode is enabled.

> exitcode

```
$exitcode = exitcode $shellcode
```

Executes on the remote host the code provided as a string. Sets the exit code of the last command executed in the $? variable. Returns the exitcode of the last command executed. The provided parameter will be displayed if debug mode is enabled.

> script

```
$output = script pathToLocalScript

( $output, $exitcode ) = script pathToLocalScript
```

Executes on the remote host the provided local script. Sets the exit code of the last command executed in the $? variable. In scalar context returns the output of the execution as a string. In list context returns the output of the execution and the exit code of the last command executed as a two item list.

+++ Stateful +++

> file

```
$success = file remotePath, %options
```

Executes a remote path(s) or file(s) operation(s) on the remote host, specified by the given options. Returns true if no error was encountered.

remotePath

Can be one of the following:

- **String**

  A string that holds the desired remote path which will be subject to the specified options.

- **ArrayRef**

  One or more remote paths within an ArrayRef that will be subject to the specified options.

`%options`

A **key**=>***value*** list (hash) containing any the following valid options:

ensure =>

directory

A directory will be created for each remote path given only if absent. The execution will be halted and the error will be displayed if the exit code is not zero.

absent

Each remote path given will be deleted only if exists and the effective user has write permissions. The execution will be halted and the error will be displayed if the exit code is not zero or the remote is any of the following:

- */*
- */bin*
- */boot*
- */dev*
- */etc*
- */home*
- */lib*
- */media*
- */mnt*
- */opt*
- */sbin*
- */srv*
- */tmp*
- */usr*
- */var*
- */root*
- */proc*
- */run*
- */sys*
- */usr/bin*
- */usr/include*
- */usr/lib*
- */usr/local*
- */usr/sbin*
- */usr/share*
- */usr/src*
- */usr/local/bin*
- */usr/local/lib*
- */usr/local/share*
- */var/lib*
- */var/lock*
- */var/log*
- */var/mail*
- */var/opt*
- */var/run*
- */var/spool*
- */var/spool/mail*
- */var/tmp*

source => *pathToLocalFile*

The given local source file will be uploaded to the remote host for each remote destination provided **only if the remote file is different or absent**. The execution will be halted and the error will be displayed if the upload fails.

content => *'some content to be uploaded as a file'*

The given content (may be any data, including text) will be uploaded as a file to the remote host for each remote destination provided **only if the remote file is different or absent**. The execution will be halted and the error will be displayed if the upload fails.

on_change => `\&mySub`
on_change => `$mySub`
on_change => `sub { $restart = 1; print "example subroutine $restart" }`

The provided subroutine (CODE ref) will be executed for each remote file that was changed or absent.

owner => *'newowner'*

Warns error and returns false if a given remote was not able to take specified ownership permissions.

group => *'newgroup'*

Warns error and returns false if a given remote was not able to take specified group permissions.

mode => *'newmode'*

Warns error and returns false if a given remote was not able to take specified mode.

> pkg

```
$modified = pkg packageName1, packageName2, packageNameN..., ensure => desiredState
```

Attempts to identify the remote host "OS" and install or remove the given packages according to the desired state. Returns the number of packages installed or removed. The execution will be halted if the package can not be installed or removed. A setup warning will be displayed if the remote "Operating System" is not supported out of the box.

packageName

Can be any of the following in any order or combination:

- **String:** String containing the name of the package to be installed or removed.
- **Array:** Array containing **`packageName`** parameters.
- **ArrayRef:** ArrayRef containing **`packageName`** parameters.

desiredState

Can be one of the following:

- **present**

  Any given package that is not found within the remote host will be installed.

- **absent**

  Any given package that is found within the remote host will be removed.

> service

```
service serviceName, action

service serviceName, ensure=>desiredState
```

Attempts to identify the remote host "OS" and verifies and controls services on remote hosts. A setup warning will be displayed if the remote "Operating System" is not supported out of the box.

serviceName

String containing the name of the service to be verified and/or controlled.

action

String containing the action to be sent to the given service. Actions vary among services. Some common actions are start, stop, restart, reload, enable, disable. The execution will be halted if the service does not exist on the remote host or the action exit code returned non-zero.

desiredState

Can be one of the following:

- **started**

  An attempt to start the specified service will be commited if the service is stopped. The execution will be halted if the service does not exist on the remote host or was not enabled at boot or the service is stopped and was not started.

- **stopped**

  An attempt to stop the specified service will be commited if the service is running. A warning message will be displayed and the attempt will be skipped if the service does not exist on the remote host or the service was not disabled at boot. The execution will be halted if the service is running and was not stopped.

Properties

> timeout

```
$timeout = timeout [seconds]
```

The maximum number of *seconds* that an ssh operation will be allowed to run. The exact string 'Timed out' will be displayed in red and the execution will be halted after reaching the number of *seconds* specified. If no timeout has been provided then the default timeout of **300** *seconds* is used. Returns the current number of seconds that will trigger a timeout.

> sudo_user

```
$sudo_user = sudo_user [user]
```

Sets the user to impersonate within sudo executions. Returns the current sudo user.

Errors

timed ouT - timed out while waiting password authentication

timed Out - timed out while verifying connected socket

Timed out - timed out while waiting on an ssh operation

socket error - password error or some other login/auth error

expand

Square brackets `[` and `]` within a hostname will be expanded. Multiple square brackets will perform combination.

Two types of expansion are available:

- Range

  `chararter_start..character_end`

  `number_start..number_end`

  May be ascending or descending

- List

  `number`

  `character`

  Each list element separated by comma

Examples:

- `host[3..5]` will be expanded to: `host3 host4 host5`
- `host[4,10,11]` will be expanded to: `host4 host10 host11`
- `host[a..c]` will be expanded to: `hosta hostb hostc`
- `host[11..13,15,c,d..f]app[1,2]` will be expanded to:

```
host11app1
host11app2
host12app1
host12app2
host13app1
host13app2
host15app1
host15app2
hostcapp1
hostcapp2
hostdapp1
hostdapp2
hosteapp1
hosteapp2
hostfapp1
hostfapp2
```