Concept
Using yadtshell you can orchestrate high level operations like updating a group of hosts, as well as stopping and starting relevant and dependent services in the correct order.

The most important command is `update`. After running `update` successfully, all hosts are up-to-date and all services are up and running.

Note: In case of a problem (i.e. a command terminates with exit code != 0), yadtshell stops immediately. Subsequent calls of the same command continue where the previous call stopped.

Component URIs
- `host://<hostname>`
- `service://<hostname>/<name>`
- `artefact://<hostname>/<name>/<version>`

Examples
- `host://hostname`
- `service://hostname/tomcat6` 
- `artefact://hostname/web-application/0:1.23`

Note: Components are always host-specific.

Target Configuration
yadtshell uses a yaml file named `target` in the current working directory to define a yadt target (set of hosts), e.g.

```
host://hostname0[1..3]
```

Host Configuration
The yadt minion gets configured via *.yaml files in the `/etc/yadt.conf.d` directory; they get merged in alphanumeric order.

Note: Indented blocks have to start with 4 blanks. Do not use tabs.

```
/etc/yadt.conf.d/my-services.yaml
```

The service definition may also contain a complete component URI as string, which describes a service on another host, e.g.

```
needs_services: ['service://hostname/service ']
```

Note: This notation only allows the hostname, not the full qualified domain name. Yadtshell extracts the hostname from the fqdn as the string until the first dot.

Please see the host configuration section of the yadtshell wiki for more information.

Using the yadtshell
- `start yadtshell`
- `stop yadtshell`
- `init yadtshell`

It is possible to group your hosts within a target:

```
/target
```

This will change the way the hosts will be displayed.

```
hoSTS:
  − hostname1.spam.eggs
  − hostname2.spam.eggs
  − hostname*.spammy.eggs
  − hostname0[1..3].foo.bar
```

```
hoSTS:
  − target
```

```
hoSTS:
  − hostname1.spam.eggs hostname2.spam.eggs
  − hostname3.foo.bar hostname4.foo.bar
```

This will change the way the hosts will be displayed.
Yadtshell Commands

Use the `yadtshell` command if you prefer to execute yadtshell commands without entering the yadtshell itself:

### General Usage

- **-v** verbose
- **-dryrun** no actions executed (just logging)
- **-n** same as dryrun

### Status Information

To retrieve the status of all services and artefacts versions from the current target use:

- **fetch status**
  - `> status`

This will also perform `info`, which displays a summary of all services for each host within the current target:

- **show status**
  - `> info [--full]`

- **dump json data**
  - `> dump [uri-query0 [uri-query1 ...]]`

**Additional arguments for `dump`**:

- **attribute**
- **show-pending-updates**
- **show-current-artefacts**

- **dump raw data of all services**
  - `> dump service://`

**Note** The output of info and dump is generated using cached data.

### Hosts

To prevent others from executing commands on a host it is possible to lock the host:

- **lock host**
  - `> lock -m "message" [--force] <host_uri> [...]`

Afterwards commands can only be executed
- by you,
- from the current target directory
- on the current host.

- **lock hostname01**
  - `> lock -m "I need this" host://hostname01`

**Note** The message should reflect the reason why you are doing what you are doing and include your name as well.

### Hijacking a Locked Host

- **hijacking a locked host**
  - `> lock -m "hijacking" --force host://*`

**Note** The message should reflect the reason why you are doing what you are doing and include your name as well.

### Services

To start a service, regarding its dependencies, use:

- **start service**
  - `> start <service_uri> [<service_uri> ...]

Start all services
- `> start service://`

To stop a service and all services depending on the service:

- **stop service**
  - `> stop <service_uri> [<service_uri> ...]`

**Note** When stopping a service all services depending on this service will be stopped as well. But starting the service will not start the services depending on the service again.

If a service is currently out of order you can ignore the state of a service (e.g. assume all operations on that service are successful):

- **ignore service**
  - `> ignore -m "message" <service_uri> [...]`

**Note** The message should reflect the reason why you are doing what you are doing and include your name as well.

### Artefacts

To install updates (if there are any) and stop/start the defined services use:

- **update target**
  - `> update [<host_uri> ...] [-p <number>]`

If you only want to update artefacts without restarting services, use `updateartefact`. Take care when using this command: it is ignoring all service dependencies.

- **updateartefact**
  - `> updateartefact <artefact_uri> [...]`

To release a lock use:

- **unlock host**
  - `> unlock <host_uri> [<host_uri> ...]`

Release all your locks on all hosts:
- `> unlock host://*`