

# Инструкция по созданию дополнительных параметров групповых политик (шаблонов)

ALD Pro

Exported on 08/11/2023

## Table of Contents

1	Как работают групповые политики.....	4
2	Как создать дополнительный параметр .....	5
3	Как настроить атрибуты дополнительного параметра .....	7
4	Скрипт дополнительного параметра.....	8
4.1	Императивные инструкции шаблонизатора Jinja .....	8
4.1.1	Выполнение команд .....	8
4.1.2	Работа с переменными.....	9
4.1.3	Ветвления и циклы.....	10
4.1.4	Информация о целевой системе (grains) .....	11
4.1.5	Передача информации с мастера (pillar) .....	11
4.1.6	Создание файлов на стороне миньона.....	12
4.2	Декларативные описания SaltStack.....	13
5	Требования к скриптам и наследование параметров.....	15
6	Отладка.....	16

v 1.2.3

Групповые политики позволяют снизить стоимость управления ИТ-инфраструктурой за счет автоматического применения одинаковых настроек на больших группах компьютеров, имеющих общее целевое назначение.

Каждая политика (в терминах MS объект групповой политики) представляет из себя именованный набор параметров, в соответствии с которыми производится автоматическая настройка операционной системы и прикладного программного обеспечения. Для настройки групповой политики нужно выполнить следующие действия:

- создать новую групповую политику
- добавить конфигурационный параметр в политику (включить его)
- установить желаемые значения атрибутов для параметра
- назначить политику на подразделение, внутри которого есть целевые компьютеры или пользователи
- подождать 30 минут или перезапустить службу salt-minion, чтобы форсировать применение политики на пользовательском компьютере

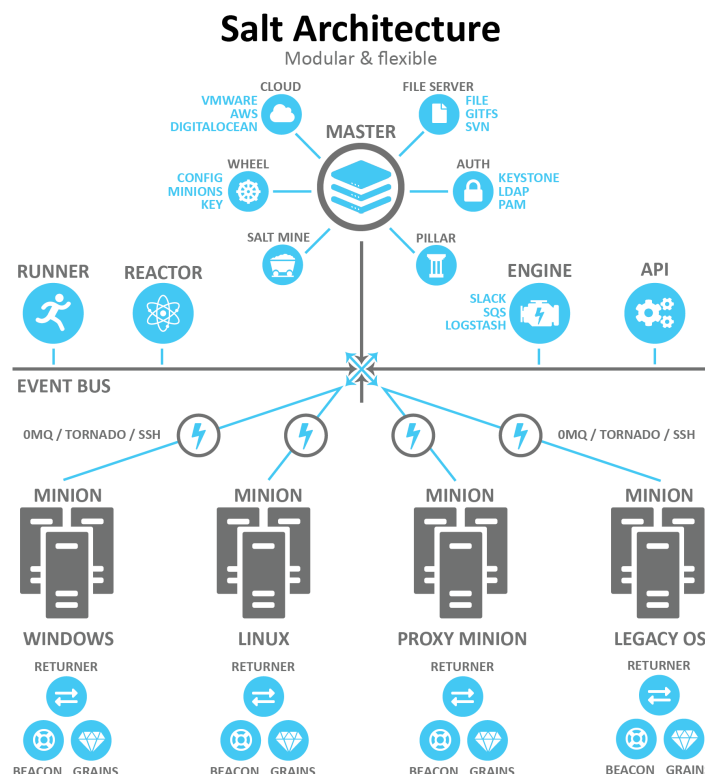
Администраторам доступны сотни параметров «из коробки», но для решения специфичных задач конкретного бизнеса может потребоваться разработка дополнительных параметров силами команды внедрения, и в ALD Pro для этого есть все необходимые инструменты.

# 1 Как работают групповые политики

В основе механизма групповых политик ALD Pro лежит SaltStack — система управления конфигурациями и удаленного выполнения операций с открытым исходным кодом, написанная на языке Python.

SaltStack работает по модели «издатель — подписчик», поэтому базовыми компонентами архитектуры являются: мастер (издатель), миньоны (подписчики) и шина данных.

- **Мастер** — это сервер, выполняющий централизованное конфигурирование рабочих станций. Данная служба работает на контроллерах домена в отказоустойчивом мультимастер-режиме, то есть политики реплицируются между контроллерами, и миньоны могут подключаться к любому из них.
- **Миньоны** — это рабочие станции или серверы, выступающие в качестве клиентов по отношению к мастеру, и принимающие от мастера команды на удаленное конфигурирование. Служба миньона запускается на всех хостах в домене, в том числе и контроллерах домена, где работает Salt Master.
- **Шина данных** — это система для передачи сообщений на базе ZeroMQ, через которую миньоны получают команды на конфигурирование и передают результаты об их исполнении. Шина работает на мастере, миньоны устанавливают постоянное соединение с шиной по порту TCP/4505 для получения заданий (порт издателя, publisher) от мастера, а для отправки отчетов о выполнении заданий периодически подключаются к нему по порту TCP/4506 (порт сервера запросов, request server). На уровне бизнес-логики применение групповых политик на миньоне возможно как в pull-режиме по инициативе миньона (см. утилиту salt-call), так и в push-режиме по инициативе мастера (см. утилиту salt и файл \_schedule.conf).



## 2 Как создать дополнительный параметр

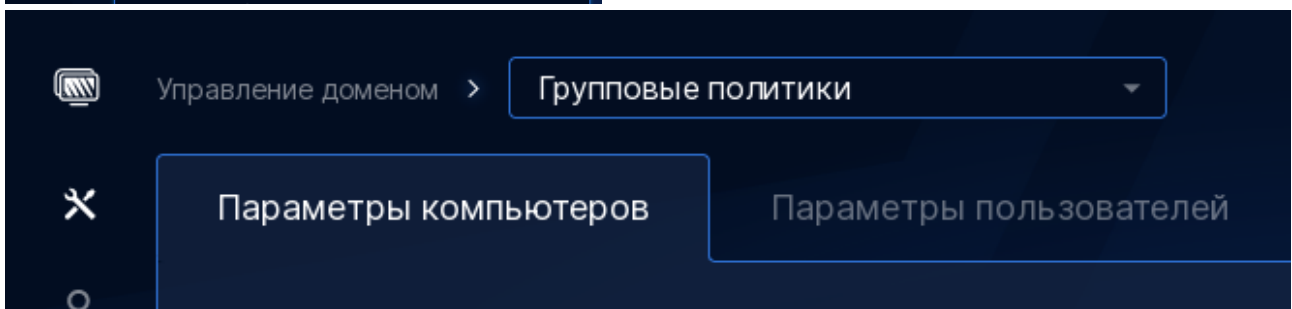
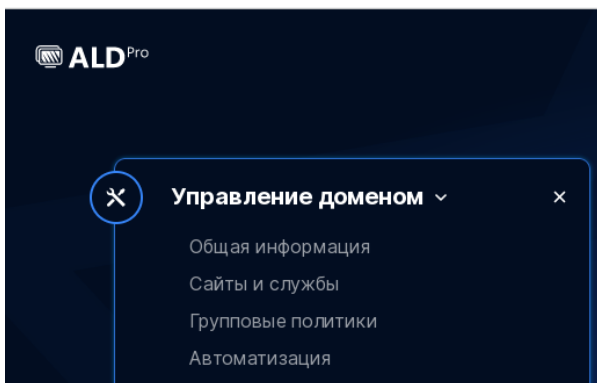
Параметр групповой политики (в терминах MS шаблон групповой политики) подобен классу из теории объектно-ориентированного программирования. Он определяет модель для создания объектов конфигурирования, описывая их свойства (атрибуты) и методы для работы с этими данными (скрипт). Продолжая аналогию из ООП, «объектом» в данном случае будет являться экземпляр параметра, добавленный в конкретную групповую политику.

Параметр может быть «простым» или «составным»:

- В случае **простого** параметра свойства представляют из себя плоский список атрибутов. Например, для настройки межсетевого экрана может потребоваться задать уровень детализации журнала, и эта настройка подразумевает одно конкретное значение, поэтому она может быть реализована атрибутом «простого» параметра.
- **Составные** параметры позволяют задавать таблицу настроек, где в каждой строке представлен однотипный набор данных. Если продолжать пример с межсетевым экраном, то для настройки фильтрации может потребоваться разрешить входящие ssh-соединения, запретить исходящие smtp и т.п., поэтому такие настройки нужно реализовывать атрибутами «составного» параметра.

Параметр может быть «параметром компьютера» или «параметром пользователя». «Параметры компьютеров» отвечают за настройку оборудования и служб вне зависимости от того, какой пользователь работает с системой. «Параметры пользователей» отвечают за настройку рабочей среды пользователя вне зависимости от того, на каком компьютере он работает. В обоих случаях скрипты выполняются с возможностями по администрированию root-пользователя.

Чтобы создать новый параметр групповой политики вам нужно перейти на страницу «Управление доменом \ Групповые политики» и выбрать одну из вкладок:



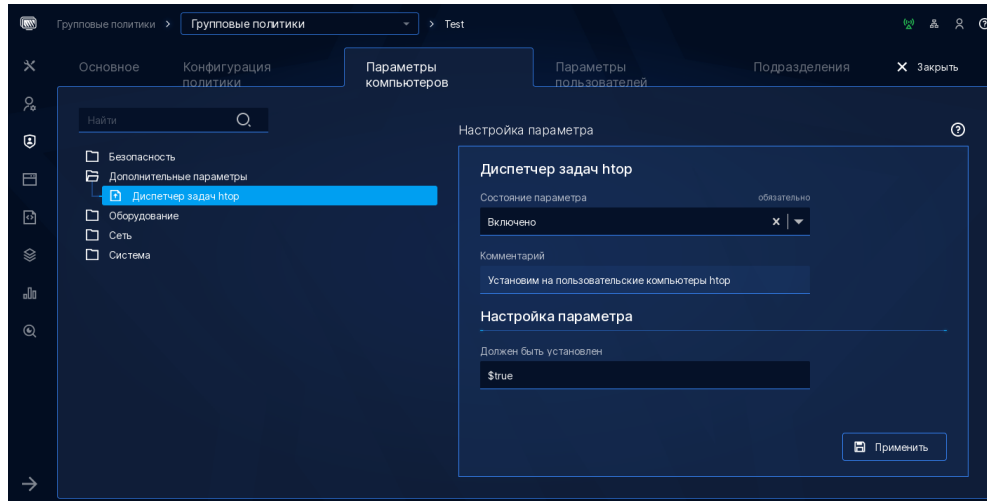
На каждой вкладке по умолчанию будет представлен корневой каталог «Дополнительные параметры», внутри которого вы можете создавать свои пользовательские параметры. Если параметров станет слишком много, вы сможете группировать их по подразделам.

Выберите раздел «Дополнительные параметры», нажмите кнопку «+ Новый параметр» и заполните карточку следующими значениями:

- Раздел «Основные»
  - Название параметра: «Диспетчер задач htop»  
*Это название, которое вы хотите видеть в каталоге при редактировании групповых политик на портале управления.*
  - Уникальный идентификатор: «software\_htop»  
*Это название, которое будет использоваться в качестве имени \*.sls файла на диске и в скриптах SaltStack.*
  - Тип каталога: «Параметр компьютерной групповой политики»  
*Определяет, относится ли данный параметр к настройкам компьютера или пользователя. Параметр не доступен для редактирования, его значение определяется автоматически в зависимости от того, с какой страницы вы перешли к созданию дополнительного параметра.*
  - Тип параметра: «простой»  
*Позволяет указать способ хранения атрибутов.*
  - Папка параметра: «Дополнительные параметры»  
*Указывает раздел каталога, в котором будет доступен данный параметр.*
  - Назначение параметра: «Управление установкой и настройкой диспетчера задач htop»  
*Текстовое описание, позволяющее указать наиболее важную информацию о работе параметра для удобства последующего использования. Данный текст отображается при редактировании групповой политике по нажатию кнопки вопроса*
- Разделы «Атрибуты параметра» и «Конфигурация скрипта» станут доступны при редактировании сохраненного параметра.

### 3 Как настроить атрибуты дополнительного параметра

Атрибуты — это свойства, которые можно задавать при добавлении параметра к групповой политике. Например, если мы создадим параметр для управления диспетчером задач htop на рабочих станциях, то нам понадобится атрибут «Должен быть установлен», чтобы в зависимости от его значения устанавливать или удалять указанное программное обеспечение.



Атрибуты представляют из себя обычные текстовые строки, значение которых может быть интерпретировано в скриптах в том числе как числа, логические значения (\$true и \$false), списки и т. п. Добавьте атрибут:

- Название атрибута: «Должен быть установлен»  
*Название, под которым вы хотите видеть этот атрибут на карточке параметра.*
- Уникальный идентификатор: «must\_be\_installed»  
*Идентификатор атрибута, который будет использоваться как имя переменной в SaltStack скриптах.*
- Описание: «атрибут используется как логическая переменная \$true/\$false для ветвления шаблона»  
*Краткие комментарии для инженера, которые помогут упростить поддержку этого атрибута в будущем, когда потребуется внести какие-либо изменения. Данная информация недоступна при настройке групповой политики, поэтому, если вы хотите дать подсказку администратору о возможных значениях этого атрибута, внесите эту информацию в описание параметра.*

## 4 Скрипт дополнительного параметра

Скрипты SaltStack похожи на PHP, в них текст перемежается императивными инструкциями языка программирования, по результату выполнения которых получается итоговый документ. За императивную часть отвечают инструкции шаблонизатора Jinja2, а декларативная часть задается по правилам SaltStack в YAML-подобном синтаксисе.

### 4.1 Императивные инструкции шаблонизатора Jinja

Инструкции языка Jinja внедряются с помощью фигурных скобок:

Jinja	Аналог PHP	Комментарий
{% ... %}	<? ... ?>	Синтаксис для вставки инструкции языка шаблонизатора Jinja
{% print(...) %}	<? echo (...) ?>	Пример инструкции для вывода в документ по месту вызова значения выражения
{{ ... }}	<?= ... ?>	Краткий синтаксис для вывода в документ по месту вызова значения выражения
{# ... #}	<?/* ... */?>	Синтаксис для вставки комментариев средствами языка Jinja2

Для повышения читабельности кода управляющие структуры обычно оформляют отступами, но yaml-документы крайне чувствительны к пробелам, поэтому данный инструмент форматирования оказывается недоступен без применения специальных операторов подавления отступов. Чтобы убрать лишние пробелы и пустые строки вам нужно всего лишь поставить знак дефиса слева, справа или с обоих концов управляющего блока:

{%-	Удаляет пробелы и пустые строки слева
-%}	Удаляет перенос текущей строки и переносит ее на предыдущую
{%- и -%}	Удаляет пробелы и пустые строки слева, в т.ч. перенос текущей строки, поэтому текст окажется в конце предыдущей строки

#### 4.1.1 Выполнение команд

Создайте файл test.sls с простейшим Jinja-скриптом, чтобы проверить, как это работает, и выполните его с помощью команды salt-call с указанными параметрами:

```
# echo '{% do salt.log.info("Hello world!") %}' > hello.sls
```



```
# salt-call --local --file-root=. state.sls hello
```

Несколько комментариев:

- Оператор «do» вызывает метод [salt.log.info](http://salt.log.info)<sup>1</sup> с параметром 'Hello world!'
- Команда salt-call отвечает за локальное исполнение скриптов на миниионах.
- Параметр local исключает обращение к мастеру для получения настроек.
- Параметр file-root со значением «.» устанавливает в качестве рабочего каталога текущую директорию.
- Параметр state.sls указывает на то, что следует использовать метод sls модуля salt.modules.state, который применяет состояния, описанные в одном и более файлах на диске из рабочей директории
- параметр hello задает имя файла hello.sls, которое должно быть указано без расширения файла

Приведем еще один пример. С помощью метода run модуля cmd вы можете выполнить на удаленном хосте любую команду bash:

```
# echo '{% do salt.cmd.run("apt-get install htop") %}' > installhtop.sls
# salt-call --local --file-root=. state.sls installhtop
```

Полный перечень всех модулей SaltStack можно найти на странице документации <https://docs.saltproject.io/en/latest/ref/modules/all/index.html>

## 4.1.2 Работа с переменными

Переменные на языке Jinja задаются оператором set. Вам доступен широкий перечень типов данных: булевы, числовые, текстовые, списки, кортежи, словари:

```
{% set boolean_val = True %}
{% set int_val = 777 %}
{% set string_val = 'hello' %}
{% set list_val = ['h', 'e', 'l', 'l', 'o'] %}
{% set tuple_val = ('h', 'e', 'l', 'l', 'o') %}
{% set dict_val = { b: True, i: 777, s: 'hello' } %}
```

При работе с числовыми переменными доступны обычные математические операторы:

```
{% set a = 5 %}
{% set b = 10 %}
{% set a = a + b %}
{% set b = a - b %}
{% set a = a - b %}
```

Конкатенация строк возможна как специальным оператором «~», который предварительно конвертирует все значения в строки, так и обычным оператором сложения «+»:

<sup>1</sup> <http://salt.log.info>

```
{% set a = 5 %}
{% set b = 10 %}
{% set c = a ~ b %}
{% set d = 'World' %}
{% set e = 'Hello, ' + d %}
```

Для выполнения более сложных преобразований над переменными вам доступно множество функций, которые можно применять как методы через точку или в стиле конвейеров bash через символ вертикальной черты, за что их так же называют фильтрами:

```
{% set a = 'Hello'.upper() %}
{% set b = 'world' | upper %}
```

Полный перечень встроенных фильтров Jinja можно найти в документации <https://jinja.palletsprojects.com/en/2.11.x/templates/#list-of-builtin-filters>

### 4.1.3 Ветвления и циклы

Если выполнение набора команд должно зависеть от некоторого условия, то ветвление можно задать с помощью операторов if/elseif/else/endif

```
{% if ram >= 2048 %}
...
{% elseif ram <= 4096 %}
...
{% else %}
...
{% endif %}
```

Циклы for являются аналогом конструкций типа foreach других языков программирования и предназначены для выполнения заданного набора инструкций применительно к каждому элементу из списка:

```
{% set users = ['ivanov', 'petrov', 'kuznetsov'] %}
{% for user in users %}
{% do salt.log.info(user.upper()) %}
{% endfor %}
```

Если вам нужно будет обработать данные словаря, воспользуйтесь методом items(), чтобы получить список его элементов:

```
{% set users = {'u1':'ivanov', 'u2':'petrov', 'u3':'kuznetsov'} %}
{% for id, user in users.items() %}
{% do salt.log.info(user.upper()) %}
{% endfor %}
```

Так как циклы Jinja работают только со списками, то для эмуляции обычных циклов со счетчиком вам нужно воспользоваться функцией `range([min], max)`. Если передать этой функции один параметр, то будет сгенерирован список с указанным количеством элементов, нумерация которых будет начинаться с нуля. Если передать два числа, то нумерация элементов будет в указанном диапазоне.

```
{% do salt.log.info('Обратный отсчет') %}
{% for i in range(0, 10) %}
{% do salt.log.info(10 - i) %}
{% endfor %}
```

#### 4.1.4 Информация о целевой системе (grains)

При написании скриптов Jinja вы можете опираться на информацию о целевом хосте, на котором этот скрипт будет применен. Данная информация доступна в словаре `grains`, тут вы найдете информацию об операционной системе, памяти, дисках, настройках сетевых интерфейсов и многом другом. Например, используя ключ `nodename` можно получить имя хоста:

```
{% set node = salt['grains.get']('nodename') %}
```

Актуальное содержание словаря `grains` для конкретного хоста можно посмотреть с мастера утилитой `salt`, или с миньона утилитой `salt-call`:

```
# salt pc-1.ald.company.local grains.items
# salt-call grains.items
local:
  -----
  astra_version:
  -----
  distr:
    orel
  version:
    1.7.2
  ...
```

Более подробную информацию о модулях `grains` можно найти в документации по проекту: <https://docs.saltproject.io/en/latest/ref/grains/all/index.html>

#### 4.1.5 Передача информации с мастера (pillar)

При настройке групповых политик администраторы задают значения атрибутов, которые в ALD Pro передаются через механизм пилларов (`salt pillar`, соляной столб). Актуальное содержание словаря `pillar` можно посмотреть с мастера утилитой `salt`, или с миньона утилитой `salt-call`:

```
# salt pc-1.ald.company.local pillar.items
# salt-call pillar.items
local:
```

```

-----
aldpro-hosts:
  -----
  pc-1.ald.company.local:
    -----
    rbta_ldap_env_vars_h:
      -----
      rbta_ldap_env_vars_h__variables:
        |_
          -----
          rbta_ldap_env_vars_h__variables__description:
          rbta_ldap_env_vars_h__variables__important:
          rbta_ldap_env_vars_h__variables__name:
            testvar
          rbta_ldap_env_vars_h__variables__value:
            ok
aldpro-software:
  -----
  pc-1.ald.company.local:
    -----
  ...

```

Содержание словаря определяется тем, какие политики назначены конкретному компьютеру (условный аналог GPRresult в MS AD). Данные пиллара доступны только тем минионам, для кого они предназначены, поэтому через атрибуты можно передавать в том числе конфиденциальную информацию, например пароли служебных учетных записей, сертификаты.

Если же вам нужно проверить, была ли уже применена конкретная политика на хосте, вы можете применить состояние с параметром test=True. Если по результатам выполнения команды вы увидите сообщение "No changes needed to be made", значит система уже находится в целевом состоянии:

```

# salt-call state.apply rbta_ldap_env_vars_h test=True
...
      ID: /etc/bash.bashrc
      Function: file.blockreplace
      Result: True
      Comment: No changes needed to be made
...

```

Более подробную информацию о модулях pillar можно найти в документации по проекту: <https://docs.saltproject.io/en/latest/topics/tutorials/pillar.html>

#### 4.1.6 Создание файлов на стороне миниона

Перенос обычных файлов с мастера на минионы возможен с помощью модуля состояния файловой системы salt.states.file

```

/etc/foo.conf:
  file.managed:
    - source:
      - salt://foo.conf.{{ grains['fqdn'] }}

```

```

- salt://foo.conf.fallback
- user: foo
- group: users
- mode: 644
- attrs: i
- backup: minion

```

Есть методы для создания файлов touch, переименования rename, удаления clea. Подробное описание возможностей доступно на странице

<https://docs.saltproject.io/en/latest/ref/states/all/salt.states.file.html>

<https://www.8host.com/blog/osnovy-saltstack-bazovye-termíny-i-ponyatiya/>

<https://serverspace.ru/support/help/ispolzovanie-platformy-salt/>

## 4.2 Декларативные описания SaltStack

Декларативная часть скрипта описывает в YAML-формате желаемую конфигурацию компьютера. В структуре документа указано, какие методы нужно вызывать для конфигурирования системы, и с какими параметрами. В качестве примера создадим файл software.sls со скриптом, который описывает состояние, после применения которого в системе должен стать доступен диспетчер задач httpd:

```

# cat software.sls
software_httpd: # Имя состояния
  *pkg.installed: # Вызов метода installed модуля pkg
  *--*pkgs: # Аргументы метода installed
  *--*httpd # Значение аргумента pkgs

```

Вы можете выполнить скрипт с помощью команды salt-call:

```

# salt-call --local --file-root=. state.sls software

```

Рассмотрим параметры команды подробнее:

- local — означает локальную обработку без обращения к мастеру
- file-root=. — устанавливает рабочую директорию, в которой будет выполняться поиск \*.sls файлов
- state.sls — указывает, что следует использовать модуль sls, который отвечает за применение состояний из файлов на диске
- software — указывает имя \*.sls файла без расширения

Язык YAML, также как и JSON, является языком разметки текста для сериализации данных. Каждая строка задает пару ключ-значение, между которыми стоит знак двоеточия. Ключи могут состоять из одного и более слов, причем, заключать их в кавычки необязательно. В качестве значений поддерживаются как скалярные типы данных (int, float, boolean, string), так и вложенные словари, что позволяет создавать древовидные структуры данных. Второй и последующий уровни дерева должны обозначаться отступами с помощью двух и более пробелов, использовать символы табуляции вместо пробелов недопустимо. Если требуется прокомментировать какой-то фрагмент документа, то слева от комментария нужно поставить символ решетки.

На верхнем уровне структуры данных должен быть указан ключ с именем состояния, в нашем случае это `software_htop`. Имя выбирается произвольно, в одном документе может быть описано несколько состояний, но их имена не должны повторяться.

На второй строке указано, что требуется вызвать метод `installed` модуля `salt.states.pkg`. Данный модуль является модулем состояния, т. е. его методы приводят систему к требуемому состоянию, описывая желаемый конечный результат, а не то, как к нему прийти. Есть также модули исполнения, которые выполняют в системе конкретные действия, ранее мы уже показывали пример с использованием метода `run` модуля `cmd` для выполнения `bash` команды.

На третьей и четвертой строке методу передается аргумент `pkgs` со значением `htop`. Обратите внимание, что аргумент и его значение являются элементами списков, поэтому в начале этих строк стоит символ дефиса.

## 5 Требования к скриптам и наследование параметров

В соответствии с архитектурой ALD Pro скрипты групповых политик выполняются на всех хостах, вне зависимости от того, назначен этот параметр компьютеру или нет, поэтому в скрипте должна быть соответствующая проверка, используя данные из пиллара. Например, в следующем примере установка диспетчера задач будет выполнена только в том случае, если компьютер входит в подразделение, на которое назначена групповая политика с дополнительным параметром `software_htop`:

```
{% set gpo_name = 'software_htop' %}
{% set node = salt['grains.get']('nodename') %}
{% set gpo = salt['pillar.get']('aldpro-hosts:' + node + ':' + gpo_name) %}
{% if gpo %}
{{ gpo_name }}:
  ··pkg.{% if gpo.must_be_installed == '$true' %}installed{% else %}removed{% endif %}:
  ······pkg:
  ······-·htop
{% endif %}
```

Если на один компьютер назначено несколько политик, использующих один и тот же параметр, то значения простых параметров будут переопределяться, а значения составных параметров суммироваться. Для понимания этого механизма следует рассмотреть, как групповые политики применяются на конкретном хосте. ALD Pro анализирует политики, назначенные на компьютер, снизу вверх по дереву подразделений и формирует единый словарь `pillar`. Если на одно подразделение назначено несколько политик, их атрибуты обрабатываются в порядке указанных приоритетов. Скрипты групповых политик отработывают на компьютере только один раз с итоговым содержанием словаря `pillar`, вне зависимости от того, сколько раз соответствующий параметр был использован в групповых политиках, назначенных на этот компьютер.

## 6 Отладка

Скрипт дополнительного параметра сохраняется на мастере в папке `/etc/salt/states/aldpro/policies/software_htop/init.sls` и при выполнении кешируется на миниионе в папке `/var/cache/salt/minion/files/base/policies/software_htop/init.sls`.

В соответствии с расписанием из файла `/etc/salt/minion.d/_schedule.conf` скрипты выполняются при запуске службы `salt-minion` и по расписанию каждые 30 минут (1800 секунд, см. задание `update_gpo`). Посмотреть текущие задания планировщика можно командой:

```
# salt.call schedule.list
```

Чтобы не ждать так долго для проверки групповых политик можно перезапустить службу вручную (аналог `gputdate /force`):

```
# systemctl restart salt-minion.service
```

Но для отладки конкретного скрипта удобнее использовать метод `state.apply` и запускать его на стороне минииона в режиме отладки:

```
# salt-call -l debug state.apply имя_параметра
```

Через `salt-call` можно эмулировать повторное применение стейта `update_gpo`, как это происходит при запуске службы и по заданию планировщика. Такой способ может дать дополнительную информацию для отладки:

```
# salt-call -l debug state.apply update_gpo
```

Внутри SaltStack скрипта можно делать вывод сообщений в лог для трассировки значений переменных. При запуске скрипта на стороне минииона данная информация будет видна прямо из консоли, а если запускать скрипт на стороне мастера, то эти сообщения можно будет найти только в журнале на миниионе `/var/log/salt/minion`. Но учтите, что шаблонизатор отрабатывает задолго до применения стейтов, поэтому возможны ситуации, когда записи в журнале есть, а стейты не применяются из-за ошибок.

```
{% do salt.log.info("myvar = " + myvar) %}
```