# Использование скрипта сбора конфигураций

Данная инструкция предназначена для запуска скрипта на ОС Windows версии 10, проверена на 20.01.2023.

## Установка и подготовка окружения

1. Если python не установлен, или стоит версия старше 3.7, нужно скачать Python поддерживаемых версий (3.7-3.9). Версия 3.9 качается по ссылке:  
   <https://www.python.org/ftp/python/3.9.0/python-3.9.0rc2-amd64.exe>
2. Установить python 3.9, при установке поставить галку "Add python to path", остальные настройки "по умолчанию".
3. Скачать git по ссылке:  
   <https://github.com/git-for-windows/git/releases/download/v2.39.1.windows.1/Git-2.39.1-64-bit.exe>   
   и установить Git. При установке выбрать редактор в качестве дефолтного для Git из установденных в системе (например, notepad). Остальные настройки по умолчанию.
4. Создать папку, в которой будут запускаться скрипты, например, C:\scripts.
5. В контекстном меню в папке (правая кнопка мыши в проводнике) выбрать "Git Bash Here". Запустится окно терминала git.
6. Наберите команду:

> git clone https://github.com/asha77/config\_collector.git

Эта команда скачает скрипт в папку config\_collector.

Перейдите в эту папку:

> cd config\_collector

Не закрывайте терминал.

1. Далее нужно окружение и учетную запись, с которой будет осуществляться подключение к устройствам.

Все команды, выполняемые скриптом, относятся к категории show, они не требуют повышенных привилегий для работы. Может использоваться учетная запись c уровнем privilege 0, пароль для повышения привилений (*AUTH\_SECONDARY*) указывать не требуется.

Для настройки окружения:

* в проводнике сделать копию файла ".env\_init" и переименовать ее в ".env". Если проводник не показывает расширения файлов, это можно включить в настройках проводника.
* открыть файл ".env" в notepad и изменить значения переменных AUTH\_USERNAME, AUTH\_PASSWORD, чтобы файл выглядел так:

*AUTH\_USERNAME=<указать имя пользователя с которым можно авторизоваться на устройствах>*

*AUTH\_PASSWORD=<указать пароль этого пользователя>*

*AUTH\_SECONDARY=*

*AUTH\_STRICT\_KEY=False*

*TRANSPORT=sshv2*

*TIMEOUT\_SOCKET=5*

*TIMEOUT\_TRANSPORT=10*

*WORKING\_DIRECTORY=*

**Любые пробелы, табуляция после '=' или значений ключей должна отсутствовать.**

Файл ".env" будет храниться **только локально** на вашей машине и не будет синхронизироваться с репозиторием, т.к. не существует в репозитории и добавлен в “.gitignore”.

1. Создать файл 'devices.txt' на базе примера в репозитории и сохранить этот файл в корень папки config\_collector.

Аналогично .env файлу, он будут храниться **только локально** и не будут синхронизироваться с репозиторием, т.к. не существуют в репозитории и добавлены в “.gitignore”. При необходимости выполнить команды на других устройствах и/или другой набор команд, файл редактируется локально.

1. Состав команд для Cisco, которые запланированы к выполнению, приведен в файле «cisco\_commands.txt» в корне папки config\_collector. Он заведомо избыточен: команды нерелевантные для того или иного оборудования просто вызовут сообшение в консоли устройства и далее будут проигнорированы.

Команды п.10-13 выполняются последовательно в терминале git, который мы не закрывали на шаге 6.

1. В консоли git в папке config\_collector запустить команду создания виртуального окружения python.

> python -m venv venv

1. Активировать виртуальное окружение python:

> source ./venv/scripts/activate

1. Обновить менеджер пакетов pip:

> python.exe -m pip install --upgrade pip

1. Установить необходимые библиотеки:

> pip install -r requirements.txt

## Запуск скрипта

1. После успешного завершения процесса установки окружения, можно запустить YAUCC командой в терминале git:

>./start.sh

Скрипт сначала активирует venv, а затем запускает YAUCC c параметрами:

python -u -O main.py -d devices.txt

1. В ходе работы скрипт пытается:

* подключиться к каждому устройству с логином и паролем пользователя, заданном в файле .env.
* получить результат команды 'show ver', чтобы определить доступность устройства и его тип.
* После первичного опроса, скрипт заново подключается к распознанным устройствам и выполняет на каждом устройстве команды из файла "commands.txt". По мере опроса устройств скрипт выводит в консоль и лог-файл информационные сообщения.

1. Результаты работы скрипта сохраняются в папки:

\config\_collector\output\cnf\_<data и время>\

При каждом запуске скрипта создается новая папка. После завершения работы скрипта папки можно архивировать или удалять.

1. После завершения работы скрипта (для большого количества команд опрос занимает до 90 секунд на устройство) python закончит выдаст сообщение о завершении работы.
2. Результат работы скрипта (собранные данные) хранятся в последней по времени папке с результатами:

\config\_collector\output\cnf\_<data и время>\

Эту папку необходимо заархивировать, закрыть архив паролем и можно передавать дальше.

## Обновление скрипта

Производится в консоли git командой:

> git pull

В результате git скачает изменения кода скрипта, при этом никаких изменений и настроек для обновленной версии производить не нужно (если это явно не указано).

## Действия при возникновении ошибок

Если в процессе работы скрипта наблюдались ошибки, стоит посмотреть результат сбора, хотя бы выборочно) – если в целом файлы с результатами содержат команды и ответы устройства, то скорее всего все нормально.

Если в результате совсем ничего не получилось собрать, следует сделать следующее:

* Запустить еще раз скрипт из консоли git командой:

>./start\_debug.sh

* дождаться завершения его работы (он соберет больше диагностических данных).
* файл 'logfile.txt' из последней по времени папки c результатами работы программы.
* файл scrapli.log из папки config\_collector.
* вывод или скрипшот команд проверки версий ПО из консоли git:

> python --version

> pip --version

> pip list