ДЗ #01. Linux & Tmux

Дедлайны

Мягкий deadline: 12.10, 23:59
Жесткий deadline: 19.10, 23:59

Основная идея

Вам необходимо написать консольную программу с именем **`homework`**, которая будет управлять набором запущенных копий веб-приложения на Python **server.py**.
Функционал вашей программы:
1. Запустить новую копию веб-сервера:

```
./homework start --name <server_name> --port <server_port>
```

где **`<server_name>`** — уникальное имя новой копии сервера, **`<server_port>`** — порт, на котором вам нужно запустить новую копию сервера. Уникальное имя состоит исключительно из букв в нижнем регистре длиной от 1 до 32 символов.

Запуск новой копии сервера состоит в следующем:

Создать в текущей папке новую папку с именем **`<server_name>`**
Скопировать в новую папку файл веб-приложения **`server.py`**, он будет находиться в той же директории, из которой будем запускать ваше приложение.
Запустить копию веб-приложения **в новой папке внутри tmux-сессии с именем homework в отдельном tmux-окне** с именем **`<server_name>`** следующим образом:

```
python server.py --port <server_port>
```

Копия должна быть запущена именно из своей папки!

Примечание: порядок аргументов **`--name <server_name>`** и **`--port <server_port>`** может быть произвольный!

Для продвинутого потока:

Дополнительно нужно реализовать проверки:

Если существует копия сервера с именем `<server_name>` — `exitcode 1`
Если существует копии сервера с портом <server_port> — `exitcode 2`
порт <server_port> занят процессом, отличным от сервера — `exitcode 3`
после выполнения команды для запуска копии сервера подождать 3 секунды и проверить его работоспособность с помощью запроса к копии сервера:

```
http://localhost:<server_port>/healthcheck
```

если ответ имеет статус-код, отличный от 200, то вывести этот статус-код в stderr в формате:

```
Bad response code: <response_code>
```

а также завершить работу с `exitcode 4`

2. Остановить копию веб-сервера по имени:

`./homework stop --name <server_name>`

где **<server_name>** — уникальное имя копии сервера, который нужно остановить.
В результате выполнения этой команды:

Процесс копии веб-сервера должен перестать работать
Содержимое файла `out.txt` останавливаемого сервера должно быть перенесено в отдельную папку `.backup` (соседнюю по отношению к папке с сервером) и названо как `out_<server_name>_<stop_timestamp>.txt` , где `<stop_timestamp>` — timestamp в момент остановки копии.
Примечание — проверять timestamp будем приблизительно.
Tmux-окно этой копии должно быть закрыто
Папка этой копии должна быть удалена
Никакие другие копии не должны быть затронуты!

3. Остановить все копии веб-сервера:

`./homework stop_all`

В результате выполнения этой команды:

Все процессы копий должны быть остановлены
Tmux-окна всех копий должны быть закрыты
Tmux-сессия homework должна быть закрыта
Содержимое файлов `out.txt` должно быть забэкаплено — см. выше
Все папки копий должны быть удалены

4. Сбор результатов работы всех копий

`./homework collect_all`

В результате своей работы каждая копия сохраняет свои результаты в файл **out.txt** в своей директории. Вам необходимо собрать и вывести содержимое файлов **out.txt** всех активных копий, отсортированное по имени копии лексикографически, например, сначала

все содержимое копии `a`, затем копии `ab`, затем копии `b` и т.д. Формат вывода следующий:

```
=== server: a ===
<содержимое a/out.txt>
=== server: ab ===
<содержимое ab/out.txt>
=== server: b ===
<содержимое b/out.txt>
…
```

Для продвинутого потока:
Дополнительно ваша программа должна поддерживать опцию `--zip`, которая вместо вывода в stdout будет собирать все файлы `<server_name>/out.txt` и упаковывать их в zip-архив с именем `out_<collect_timestamp>.zip`.
Внутри архива файл с именем `out_<server_name>.txt` должен иметь такое же содержимое, как и файл `<server_name>/out.txt`. При этом исходные файлы `<server_name>/out.txt` должны остаться неизменными!

О языке и зависимостях

Саму программу можете писать либо на **Bash**, либо на **Python 3.12**.

Важно

Если ваше решение требует дополнительные зависимости (системные пакеты и библиотеки), то необходимо указать команды для их установки в скрипте `install.sh`.

Полезные советы

`libtmux` - библиотека python для работы с tmux (в среде проверки будет установлена версия 0.46.2)
`argparse` - библиотека для парсинга аргументов командной строки
`click` - альтернативная `argparse` (в среде проверки будет установлена версия `8.3.0`)
Можете создавать файлы и папки для получения желаемого результата (в рамках разумного)

Критерии оценивания

Скрипт запускается, сервер работает +0.2 (блокирующее)
tmux окна правильно названы, папки созданы +0.3 (блокирующее)
Остановка выполняется корректно +0.1
Backup log создается +0.1
Корректно отрабатывает остановка всех копий +0.15
Корректно отрабатывает сбор результатов +0.15

Формат сдачи

Выполняете задание
Загружаете выполненное задание с помощью `git` (не веб-интерфейса!) на gitlab.atp-fivt.org в свой репозиторий по заданию 1 (`<login>-hw1`) **в ветку dev** (логины скоро пришлем на ваши почты).
Создаете Merge request из ветки **dev** в ветку **main** — **не выставляйте** при этом Assignee и прочие позиции!
Проверяете, что в табличке в столбце **Queue** появилась ссылка на ваш репозиторий.
Внимание - табличка обновляется раз в 10 минут, ваш репозиторий может появиться не сразу!
Ждете, пока ваше задание одобрит проверяющий.
Если проверяющий не обнаружит проблем в вашем решении, то он выставляет баллы по вашему заданию и выполняет слияние веток.

Если проверяющий обнаружил проблемы в вашем решении, то он указывает их в комментарии к решению (вы получите уведомление на почту о новом комментарии к вашему MR). После чего вы вносите исправления и снова загружаете их в репозиторий в ту же ветку **dev**. Так продолжается, пока проверяющий не согласится принять ваше задание.