Шаблон отчёта по лабораторной работе

Простейший вариант

Дмитрий Сергеевич Кулябов

Содержание

1 Цель работы **5**

2 Задание **6**

3 Теоретическое введение **7**
3.1 Менеджер паролей pass 7
3.1.1 Основные свойства 7
3.1.2 Структура базы паролей 7
3.1.3 Реализации 8
3.2 Управление файлами конфигурации 11
3.2.1 Обшая информация 11
3.2.2 Конфигурация *chezmoi* 11
3.2.3 Шаблоны 13
3.2.4 Переменные шаблона 18

4 Выполнение лабораторной работы **19**

5 Выводы **20**

Список литературы **21**

Список иллюстраций

4.1 Название рисунка 19

Список таблиц

1 Цель работы

Создать и соединять репозиторий с утилитами, используя chezmoi.

2 Задание

установить утилиты, настроить интерфейс и подключить репозиторий к своей системе.

3 Теоретическое введение

3.1 Менеджер паролей pass

- Менеджер паролей pass — программа, сделанная в рамках идеологии Unix.
- Также носит название стандартного менеджера паролей для Unix (The standard Unix password manager).

3.1.1 Основные свойства

- Данные хранятся в файловой системе в виде каталогов и файлов.
- Файлы шифруются с помощью GPG-ключа.

3.1.2 Структура базы паролей

- Структура базы может быть произвольной, если Вы собираетесь использовать её напрямую, без промежуточного программного обеспечения. Тогда семантику структуры базы данных Вы держите в своей голове.
- Если же необходимо использовать дополнительное программное обеспечение, необходимо семантику заложить в структуру базы паролей.

3.1.2.1 Семантическая структура базы паролей

- Рассмотрим пользователя user в домене example.com, порт 22.

- Отсутствие имени пользователя или порта в имени файла означает, что любое имя пользователя и порт будут совпадать:

```
example.com.pgp
```

- Соответствующее имя пользователя может быть именем файла внутри каталога, имя которого совпадает с хостом. Это полезно, если в базе есть пароли для нескольких пользователей на одном хосте:

```
example.com/user.pgp
```

- Имя пользователя также может быть записано в виде префикса, отделенного от хоста знаком @:

```
user@example.com.pgp
```

- Соответствующий порт может быть указан после хоста, отделённый двоеточием (:):

```
example.com:22.pgp
example.com:22/user.pgp
user@example.com:22.pgp
```

- Эти все записи могут быть расположены в произвольных каталогах, задающих Вашу собственную иерархию.

3.1.3 Реализации

3.1.3.1 Утилиты командной строки

- На данный момент существует 2 основных реализации:
  - pass — классическая реализация в виде shell-скриптов (https://www.passwordstore.or

  - gopass — реализация на go с дополнительными интегрированными функциями (https://www.gopass.pw/).

- Дальше в тексте будет использоваться программа pass, но всё то же самое можно сделать с помощью программы gopass.

3.1.3.2 Графические интерфейсы

1. qtpass

   - qtpass — может работать как графический интерфейс к pass, так и как самостоятельная программа. В настройках можно переключаться между использованием pass и gnupg.

2. gopass-ui

   - gopass-ui — интерфейс к gopass.

3. webpass

   - Репозиторий: https://github.com/emersion/webpass
   - Веб-интерфейс к pass.
   - Написано на golang.

3.1.3.3 приложения для Android

1. Password Store

   - URL: https://play.google.com/store/apps/details?id=dev.msfjarvis.aps
   - Репозиторий с кодом: https://github.com/android-password-store/Android-Password-Store
   - Документация: https://android-password-store.github.io/docs/
   - Для синхронизации с git необходимо импортировать ssh-ключи.
   - Поддерживает разблокировку по биометрическим данным.
   - Для работы требует наличия OpenKeychain: Easy PGP.

2. OpenKeychain: Easy PGP
    - URL: https://play.google.com/store/apps/details?id=org.sufficientlysecure.keychain
    - Операции с ключами pgp.
    - Необходимо будет импортировать pgp-ключи.
    - Не поддерживает разблокировку по биометрическим данным. Необходимо набирать пароль ключа.

3.1.3.4 Пакеты для Emacs

1. pass
    - Основной режим для управления хранилищем и редактирования записей.
    - Emacs. Пакет pass
    - Репозиторий: https://github.com/NicolasPetton/pass
    - Позволяет редактировать базу данных паролей.
    - Запуск

        ```
        M-x pass
        ```

2. helm-pass
    - Интерфейс helm для pass.
    - Репозиторий: https://github.com/emacs-helm/helm-pass
    - Запуск

        ```
        M-x helm-pass
        ```

    - Выдаёт в минибуфере список записей из базы паролей. При нажатии Enter копирует пароль в буфер.
3. ivy-pass
    - Интерфейс ivy для pass.
    - Репозиторий: https://github.com/ecraven/ivy-pass

3.2 Управление файлами конфигурации

- Использование chezmoi для управления файлами конфигурации домашнего каталога пользователя.

3.2.1 Обшая информация

- Сайт: https://www.chezmoi.io/
- Репозиторий: https://github.com/twpayne/chezmoi

3.2.2 Конфигурация *chezmoi*

3.2.2.1 Рабочие файлы

- Состояние файлов конфигурации сохраняется в каталоге ~/.local/share/chezmoi
- Он является клоном вашего репозитория dotfiles.
- Файл конфигурации ~/.config/chezmoi/chezmoi.toml (можно использовать также JSON или YAML) специфичен для локальной машины.
- Файлы, содержимое которых одинаково на всех ваших машинах, дословно копируются из исходного каталога.
- Файлы, которые варьируются от машины к машине, выполняются как шаблоны, обычно с использованием данных из файла конфигурации локальной машины для настройки конечного содержимого, специфичного для локальной машины.
- При запуске chezmoi apply

вычисляется желаемое содержимое и разрешения для каждого файла, а затем вносит необходимые изменения, чтобы ваши файлы соответствовали этому состоянию. - По умолчанию chezmoi изменяет файлы только в рабочей копии.

3.2.2.2 Автоматически создавать файл конфигурации на новой машине

- При выполнении chezmoi init также может автоматически создать файл конфигурации, если он еще не существует.
- Если ваш репозиторий содержит файл с именем .chezmoi.$FORMAT.tmpl, где $FORMAT есть один из поддерживаемых форматов файла конфигурации (json, toml, или yaml), то chezmoi init выполнит этот шаблон для создания исходного файла конфигурации.
- Например, пусть ~/.local/share/chezmoi/.chezmoi.toml.tmpl выглядит так:

```
{{- $email := promptStringOnce . "email" "Email address" -}}


[data]
 email = {{ $email | quote }}
```

  - При выполнении chezmoi init будет создан конфигурационный файл ~/.config/chezmoi/chezmoi.toml.
  - promptStringOnce — это специальная функция, которая запрашивает у пользователя значение, если оно еще не установлено в разделе data конфигурационного файла.
- Чтобы протестировать этот шаблон, используйте chezmoi execute-template с флагами –init и –promptString, например:

```
chezmoi execute-template --init --promptString email=me@home.org < ~/.loca
```

3.2.2.3 Пересоздание файл конфигурации

- Если вы измените шаблон файла конфигурации, chezmoi предупредит вас, если ваш текущий файл конфигурации не был сгенерирован из этого шаблона.

- Вы можете повторно сгенерировать файл конфигурации, запустив:

```
chezmoi init
```

3.2.3 Шаблоны

3.2.3.1 Общая информация

- Шаблоны используются для изменения содержимого файла в зависимости от среды.
- Используется синтаксис шаблонов Go.
- Файл интерпретируется как шаблон, если выполняется одно из следующих условий:
  - имя файла имеет суффикс .tmpl;
  - файл находится в каталоге .chezmoitemplates.

3.2.3.2 Данные шаблона

- Полный список переменных шаблона:

```
chezmoi data
```

- Источники переменных:
  - файлы .chezmoi, например, .chezmoi.os;
  - файлы конфигурации .chezmoidata.$FORMAT. Форматы (json, jsonc, toml, yaml) читаются в алфавитном порядке;
  - раздел data конфигурационного файла.

3.2.3.3 Способы создания файла шаблона

- При первом добавлении файла передайте аргумент –template:

```
chezmoi add --template ~/.zshrc
```

- Если файл уже контролируется chezmoi, но не является шаблоном, можно сделать его шаблоном:

```
chezmoi chattr +template ~/.zshrc
```

- Можно создать шаблон вручную в исходном каталоге, присвоив ему расширение .tmpl:

```
chezmoi cd
$EDITOR dot_zshrc.tmpl
```

- Шаблоны в каталоге .chezmoitemplates должны создаваться вручную:

```
chezmoi cd
mkdir -p .chezmoitemplates
cd .chezmoitemplates
$EDITOR mytemplate
```

3.2.3.4 Редактирование файла шаблона

- Используйте chezmoi edit:

```
chezmoi edit ~/.zshrc
```

- Чтобы сделанные вами изменения сразу же применялись после выхода из редактора, используйте опцию –apply:

```
chezmoi edit --apply ~/.zshrc
```

3.2.3.5 Тестирование шаблонов

- Тестирование с помощью команды chezmoi execute-template.
- Тестирование небольших фрагментов шаблонов:

```
chezmoi execute-template '{{ .chezmoi.hostname }}'
```

- Тестирование целых файлов:

```
chezmoi cd
chezmoi execute-template < dot_zshrc.tmpl
```

3.2.3.6 Синтаксис шаблона

- Действия шаблона записываются внутри двойных фигурных скобок, {{ }}.
- Действия могут быть переменными, конвейерами или операторами управления.
- Текст вне действий копируется буквально.
- Переменные записываются буквально:

```
{{ .chezmoi.hostname }}
```

- Условные выражения могут быть записаны с использованием if, else if, else, end:

```
{{ if eq .chezmoi.os "darwin" }}
```

darwin

```
{{ else if eq .chezmoi.os "linux" }}
```

linux

```
{{ else }}
```

other operating system

```
{{ end }}
```

1. Удаление пробелов

- Для удаления проблем в шаблоне разместите знак минус и пробела рядом со скобками:

```
HOSTNAME={{- .chezmoi.hostname }}
```

- В результате получим:

```
HOSTNAME=myhostname
```

2. Отладка шаблона

- Используется подкоманда execute-template:

```
chezmoi execute-template '{{ .chezmoi.os }}/{{ .chezmoi.arch }}'
```

- Интерпретируются любые данные, поступающие со стандартного ввода или в конце команды.
- Можно передать содержимое файла этой команде:

```
cat foo.txt | chezmoi execute-template
```

3. Логические операции

- Возможно выполнение логических операций.
- Если имя хоста машины равно work-laptop, текст между if и end будет включён в результат:

```
# common config
export EDITOR=vi

# machine-specific configuration
{{- if eq .chezmoi.hostname "work-laptop" }}
# this will only be included in ~/.bashrc on work-laptop
{{- end }}
```

1. Логические функции
   - eq: возвращает true, если первый аргумент равен любому из остальных аргументов, может принимать несколько аргументов;
   - not: возвращает логическое отрицание своего единственного аргумента;
   - and: возвращает логическое И своих аргументов, может принимать несколько аргументов;
   - or: возвращает логическое ИЛИ своих аргументов, может принимать несколько аргументов.
2. Целочисленные функции
   - len: возвращает целочисленную длину своего аргумента;
   - eq: возвращает логическую истину arg1 == arg2;
   - ne: возвращает логическое значение arg1 != arg2;
   - lt: возвращает логическую истину arg1 < arg2;
   - le: возвращает логическую истину arg1 <= arg2;
   - gt: возвращает логическую истину arg1 > arg2;
   - ge: возвращает логическую истину arg1 >= arg2.

3.2.4 Переменные шаблона

- Чтобы просмотреть переменные, доступные в вашей системе, выполните:

```
chezmoi data
```

- Чтобы получить доступ к переменной chezmoi.kernel.osrelease в шаблоне, используйте:

```
{{ .chezmoi.kernel.osrelease }}
```

Более подробно про Unix см. в [1–4].

4 Выполнение лабораторной работы

Описываются проведённые действия, в качестве иллюстрации даётся ссылка на иллюстрацию (рис. 4.1).

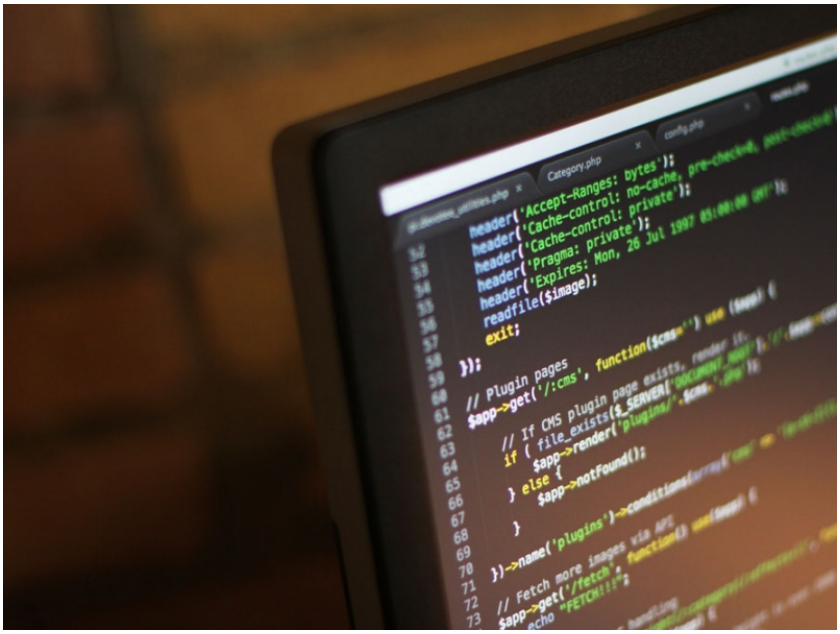

Рис. 4.1: Название рисунка

5 Выводы

Здесь кратко описываются итоги проделанной работы.

Список литературы

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.
2. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Newham C. Learning the bash Shell: Unix Shell Programming. O'Reilly Media, 2005. 354 с.