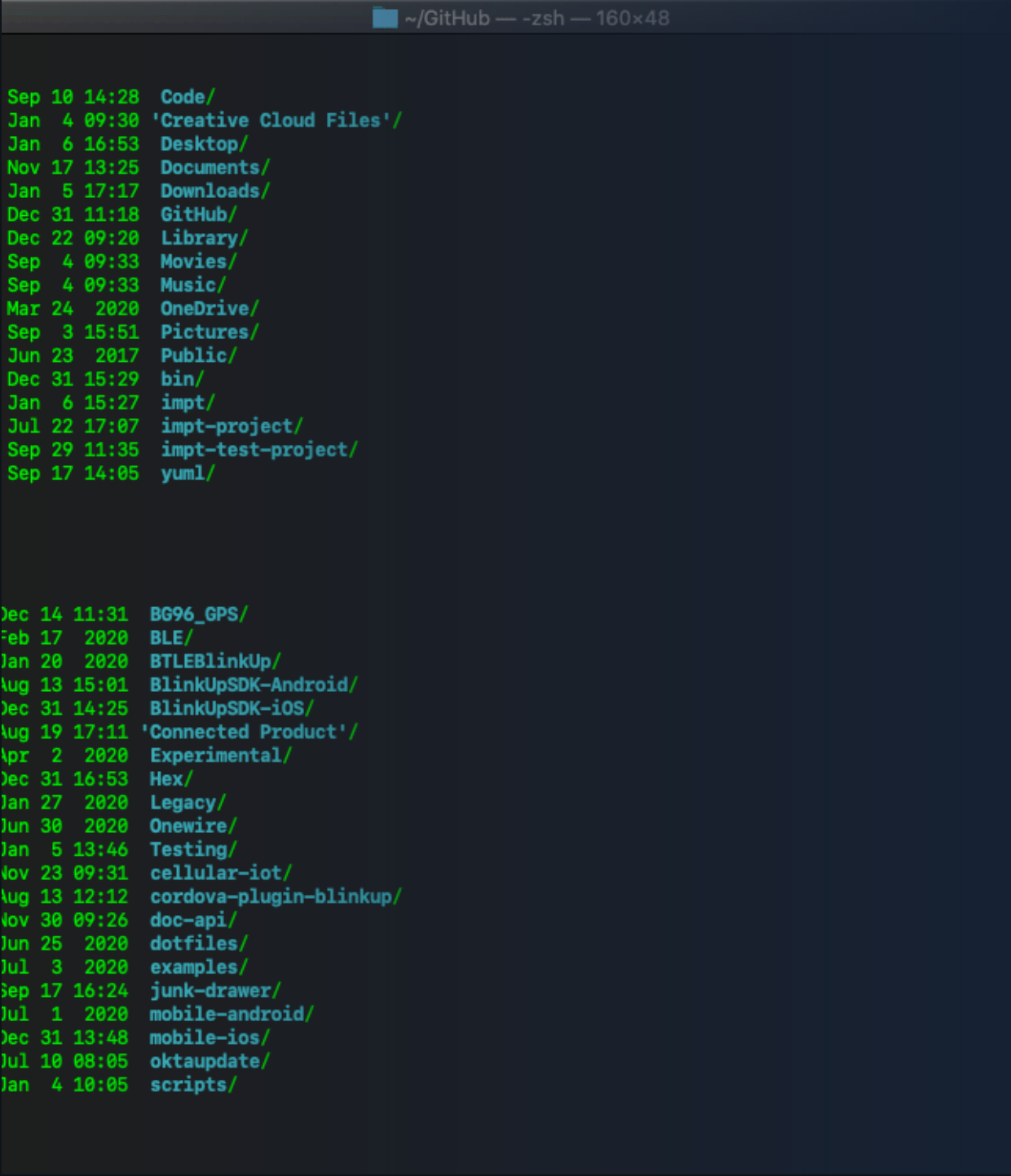
LINUX AUTOMATION

Bash-скрипти та автоматизація в Linux

Сьогодні вчимося змушувати систему працювати замість нас

Bash Cron .bashrc



Де ми зупинились?

- термінал
- grep
- awk
- пайпи

→ Ми вже обробляємо дані.
Тепер автоматизуємо.

Проблема

Що якщо команд **багато**?

Що якщо їх треба запускати **щодня**?

Рішення — скрипти

Що таке скрипт?

Текстовий файл

`.sh`

Команди

`ls, grep, awk...`

автоматизація

Скрипт — це файл, що містить послідовність команд shell, які виконуються автоматично

Перший скрипт

script.sh

```
#!/bin/bash
echo "Hello"
```

shebang (#!)

Вказує інтерпретатор для виконання скрипта

Розберемо код:

`#!/bin/bash` **Ідентифікатор скрипта**

Повідомляє систему, що це bash-скрипт

`echo` **Команда виводу**

Виводить текст у термінал

`"Hello"` **Текст для виводу**

Текст у лапках — це аргумент команди echo

Як запустити?

```
bash script.sh
```

Пряме виконання

Виконує скрипт через bash

```
chmod +x script.sh
```

Дозвіл виконання

Надає права на виконання

```
./script.sh
```

Прямий запуск

Запускає як програму

Права доступу важливі

{} Змінні

```
name="Maksim"

echo $name
```

⚠ Без пробілів!

=

Присвоєння

Змінній name присвоюється значення "Maksim"

$

Звернення

Символ $ використовується для отримання значення змінної

✕

НЕ ПИШІТЬ

```
name = "Maksim"
```

Пробіли розділяють команди!

✓

ПИШІТЬ ТАК

```
name="Maksim"
```

Без пробілів навколо знака =

Аргументи скрипта

```
echo $1

echo $2
```

$1 **Перший аргумент**
Перше значення після імені скрипта

$2 **Другий аргумент**
Друге значення після імені скрипта

Запуск скрипта:

```
./script.sh file.txt 10
```

Пояснення

`file.txt` → `$1`
`10` → `$2`

Σ Спеціальні змінні

$#

Кількість аргументів

Показує, скільки аргументів передано скрипту

$@

Всі аргументи

Містить усі аргументи, передані скрипту

$0

Ім'я скрипта

Зберігає ім'я файлу скрипта

Приклад використання

```
echo "Кількість: $#, Аргументи: $@"
```

Умова if

```
if [ -f "$1" ]; then
  echo "Exists"
fi
```

Перевірка файлу

`if` **Початок умови**

Ключове слово для початку перевірки

`[ ]` **Тестова команда**

Пара квадратних дужок для перевірки умови

`-f` **Перевірка файлу**

Повертає true, якщо файл існує

`$1` **Аргумент скрипта**

Перший аргумент, переданий скрипту

`fi` **Кінець умови**

Ключове слово для закінчення блоку if

Оператори перевірки

-f

файл

Перевіряє, чи існує **файл**

-d

директорія

Перевіряє, чи існує **директорія**

-z

порожній рядок

Перевіряє, чи є рядок **порожнім**

Приклад використання

```
if [ -f "$file" ]; then
  echo "Файл існує"
fi
```

Цикл for

```
for file in *.txt; do
  echo $file
done
```

Обробка групи файлів

`for` **Початок циклу**

Ітерується через список елементів

`file` **Змінна циклу**

Зберігає поточний елемент на кожній ітерації

`*.txt` **Маска файлів**

Всі файли з розширенням .txt у поточній директорії

`$file` **Використання змінної**

Виводить ім'я поточного файлу

`do ... done` **Тіло циклу**

Команди між do та done виконуються для кожного файлу

Цикл while

```
while read line; do
  echo $line
done < file.txt
```

Робота з файлами

`while` **Початок циклу**

Виконується поки умова істинна

`read line` **Читання рядка**

Читає по одному рядку з файлу

`$line` **Змінна рядка**

Зберігає поточний рядок з файлу

`< file.txt` **Ввід з файлу**

Перенаправляє вміст файлу в цикл

`do ... done` **Тіло циклу**

Виконується для кожного рядка файлу

Σ Функції

```
greet() {
  echo "Hello"
}
```

Структурування коду

`greet` **Ім'я функції**

Ідентифікатор, що викликає функцію

`()` **Параметри функції**

Порожні дужки — без параметрів

`{ }` **Тіло функції**

Команди між фігурними дужками

`echo` **Тіло функції**

Команди виконуються при виклику

Виклик функції

```
greet
```

Що таке змінні середовища?

Налаштування оболонки

Змінні середовища зберігають конфігурацію та налаштування для процесів в операційній системі

Шлях до домашньої директорії

`$HOME`

Шлях до виконуваних файлів

`$PATH`

Ім'я поточного користувача

`$USER`

Приклади змінних середовища

$HOME

```
echo $HOME
```

Вивід

```
/home/username
```

$PATH

```
echo $PATH
```

Вивід

```
/usr/local/bin:/usr/bin:/bin
```

$USER

```
echo $USER
```

Вивід

```
username
```

Змінні середовища використовуються для налаштування середовища виконання процесів

export

```
export NAME="Test"
```

Експорт змінної

Робить змінну доступною підпроцесам

export — **Команда експорту**

Ключове слово, що робить змінну доступною для дочірніх процесів

NAME — **Ім'я змінної**

Ідентифікатор змінної середовища (великі літери за конвенцією)

"Test" — **Значення змінної**

Значення, яке присвоюється змінній (текст у лапках)

Перехід між процесами

Батьківський процес

```
export
NAME="Test"
```

→

Дочірній процес

```
echo $NAME
```

Що таке .bashrc?

Файл налаштування shell

Налаштування виконуються при кожному відкритті нового терміналу

alias

Створення скорочень команд

змінні

Налаштування середовища

функції

Власні команди та утиліти

Для чого використовують .bashrc?

alias

```
alias ll="ls -la"
```

- Скорочення довгих команд
- Зручні псевдоніми
- Швидкий доступ до функцій

змінні

```
export PATH="$PATH:/custom"
```

- Налаштування середовища
- Змінні користувача
- Глобальні параметри

функції

```
mkcd() {
  mkdir "$1" && cd "$1"
}
```

- Власні команди
- Повторне використання коду
- Розширення функціоналу

Завдяки .bashrc ваш термінал стає зручним та персоналізованим

Приклад alias

```
alias ll="ls -la"
```

Створення скорочення

Замість `ls -la` тепер можна писати просто `ll`

`alias` **Команда alias**

Ключове слово для створення скорочення команди

`ll` **Назва скорочення**

Коротке ім'я для команди (використовуйте маленькі літери)

`=` **Присвоєння**

Оператор присвоєння зв'язує ім'я з командою

`"ls -la"` **Повна команда**

Команда, що буде виконана при використанні скорочення

Застосування змін

```
source ~/.bashrc
```

Застосування змін

Команда перезавантажує налаштування без закриття терміналу

source — Команда source

Виконує файл у поточному shell

~/.bashrc — Шлях до файлу

~ означає домашню директорію користувача

Коли використовувати

- Після редагування .bashrc
- Додавання нових alias
- Зміни змінних середовища

Різниця між файлами

.bashrc

`~/.bashrc`

- Виконується для **інтерактивних** shell
- Працює при відкритті **нового терміналу**
- Ідеальний для **alias та функцій**
- Застосовується для **налаштувань користувача**

.profile

`~/.profile`

- Виконується при **вході в систему**
- Працює один раз при **завантаженні**
- Ідеальний для **змінних середовища**
- Застосовується для **глобальних налаштувань**

.bash_profile

`~/.bash_profile`

- Виконується при **вході в bash**
- Приоритет над **.profile**
- Зазвичай викликає **.bashrc**
- Застосовується для **початкових налаштувань**

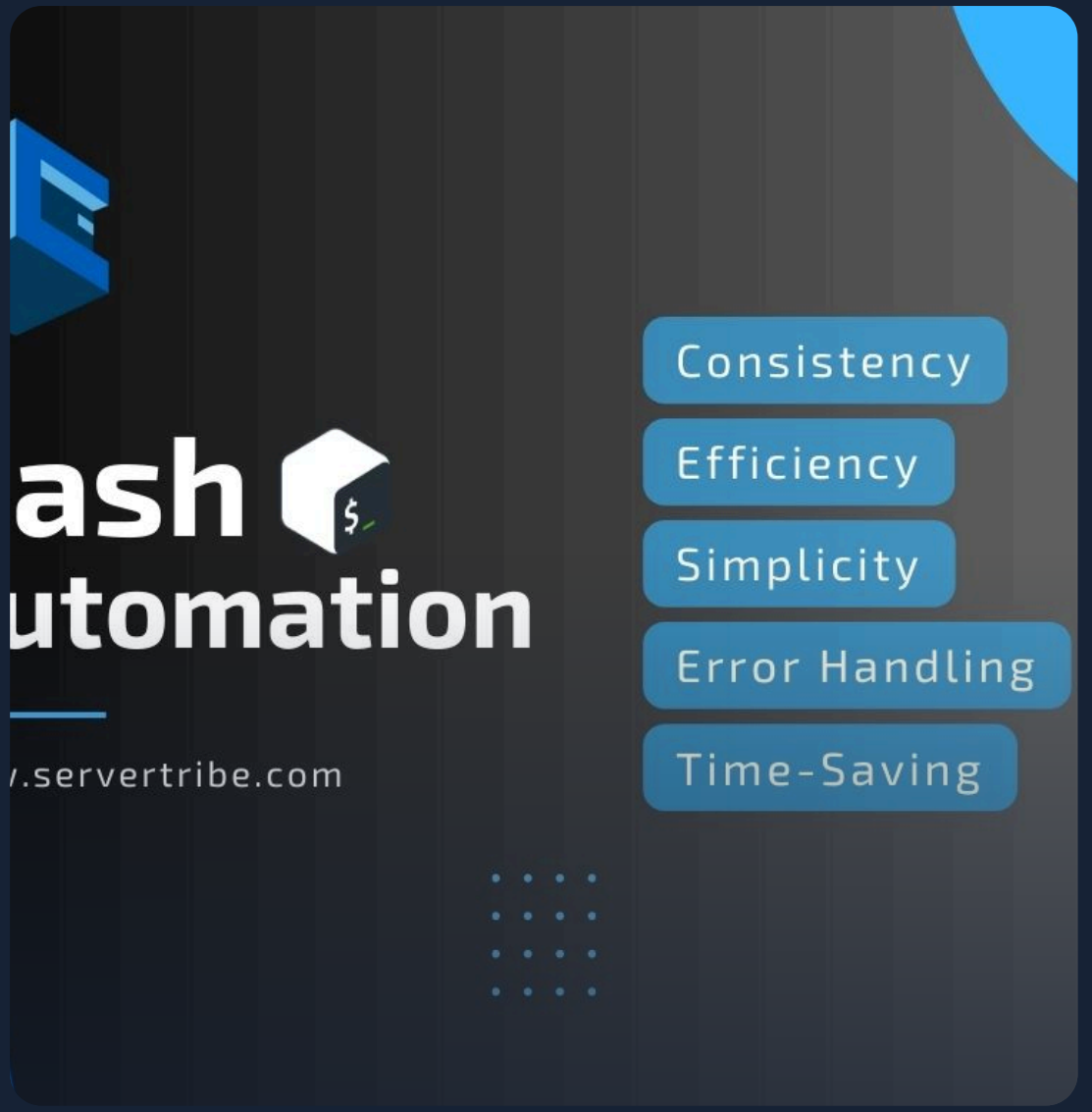
Коли що виконується — виберіть правильний файл для ваших налаштувань

Що таке cron?

Планувальник задач

Автоматичне виконання команд за розкладом

- Запуск задач за розкладом (хвилини, години, дні)
- Автоматизація повторюваних операцій
- Виконується у фоновому режимі
- Не потребує постійного підключення користувача



Формат cron

```
* * * * * command
```

5 полів часу

Кожне поле визначає одиницю часу для запуску команди

Хвилини	0-59	* * * * *
Години	0-23	* * * * *
День місяця	1-31	* * * * *
Місяць	1-12	* * * * *
День тижня	0-6 (0=неділя)	* * * * *

Поля cron

Формат

```
* * * * * command
```

Хвилини	Година	День	Місяць	День тижня
*	*	*	*	*
0-59	0-23	1-31	1-12	0-6

* означає "кожен"

Поля відділяються пробілами

Відкрити cron

```
crontab -e
```

Редагування задач

Відкриває cron-файл для редагування

`crontab`

Команда crontab

Утиліта для управління cron-задачами

`-e`

Опція -e

Відкриває cron-файл у текстовому редакторі

Як працює

- Запускає текстовий редактор (за замовчуванням - vi)
- Дозволяє додавати, редагувати та видаляти задачі
- Зберігає зміни при виході з редактора

Переглянути cron

```
crontab -l
```

Перегляд задач

Виводить список всіх cron-задач

`crontab` **Команда crontab**

Утиліта для управління cron-задачами

`-l` **Опція -l**

Виводить список всіх запланованих задач

Що показує

- Список всіх активних cron-задач
- Розклад виконання для кожної задачі
- Команди, що виконуються за розкладом

Приклад задачі

```
*/5 * * * * date >> log.txt
```

Кожні 5 хвилин
Записує дату у файл логу

`*/5`
Хвилини
Кожні 5 хвилин (0, 5, 10, 15...)

`* *`
`* *`
Інші поля
Кожна година, кожен день, кожен місяць, кожен день тижня

`date`
Команда
Виводить поточну дату та час

`>>`
Перенаправлення
Додає вивід у кінець файлу log.txt

Типові задачі

Бекап

Автоматичне створення резервних копій даних та файлів

```
0 2 * * * backup.sh
```

Очистка логів

Автоматичне видалення старих лог-файлів для звільнення місця

```
0 3 * * * clean_logs.sh
```

Моніторинг

Періодична перевірка стану системи та відправка звітів

```
*/10 * * * * monitor.sh
```

Авто-запуск

Запуск сервісів та програм при завантаженні системи

```
@reboot start_services.sh
```

Практика 1

Завдання

Створіть скрипт, який:

- Перевіряє, чи існує файл
- Створює файл, якщо його не існує
- Повідомляє про результат

Підказка

Використайте умову if та оператор перевірки -f

```
#!/bin/bash

if [ -f "file.txt" ]; then
  echo "Файл існує"
else
  touch "file.txt"
  echo "Файл створено"
fi
```

Запуск

```
chmod +x script.sh && ./script.sh
```

Практика 2

Завдання

Створіть скрипт, який:

- Приймає назву файлу як аргумент
- Рахує рядки у файлі з допомогою `wc -l`
- Фільтрує результати з допомогою `grep`
- Виводить кількість рядків

Підказка

Використайте пайпи | для поєднання команд

```
#!/bin/bash

file="$1"

wc -l "$file" | grep " "

if [ $? -eq 0 ]; then
  echo "Рядків знайдено"
fi
```

Запуск

```
./script.sh file.txt
```

Практика 3

Завдання Cron

Створіть cron-задачу, яка:

- Запускається кожні 2 хвилини
- Записує поточну дату та час у файл
- Додає записи в кінець файлу (а не перезаписує)

Підказка

Використайте формат: */2 * * * *

```
*/2 * * * * date >> cron.log
```

Кроки

1. Відкрийте crontab: `crontab -e`
2. Додайте задачу
3. Збережіть та вийдіть
4. Перевірте: `tail -f cron.log`

set -e

```
set -e
```

Зупинка при помилці

Скрипт завершується, якщо будь-яка команда повертає помилку

Як це працює

- Перевіряє статус виходу кожної команди
- При `exit code ≠ 0` зупиняє виконання
- Попереджує каскадні помилки

Приклад

```
#!/bin/bash
set -e

cd "/nonexistent"
echo "Це не виконається"
```

Скрипт зупиниться на команді cd

stderr

```
command 2> error.log
```

Обробка помилок

Перенаправляє потік помилок у файл

Потоки виводу

0	**stdin**	Ввід
1	**stdout**	Вивід (за замовчуванням)
2	**stderr**	Потік помилок

Приклад

```
ls /nonexistent 2> errors.log
```

Помилки будуть записані у файл errors.log

⇄ Bash = DevOps

CI/CD

Неперервна інтеграція та розгортання коду

Сервери

Адміністрування та управління серверами

Docker

Контейнеризація та оркестрація

Автоматизація

Автоматичне виконання рутинних задач

Це реальні робочі інструменти

Домашнє завдання

01

Скрипт-логгер

- Створіть скрипт для логування подій
- Записуйте дату та повідомлення у файл
- Додайте перевірку на існування файлу
- Зробіть скрипт виконуваним

02

Cron кожні 10 хв

- Налаштуйте cron-задачу
- Запускайте логгер кожні 10 хвилин
- Використовуйте `*/10 * * * *`
- Перевірте роботу через `crontab -l`

03

Два alias

- Додайте 2 корисних alias у `.bashrc`
- Приклад: `ll` для `ls -la`
- Застосуйте зміни через `source ~/.bashrc`
- Перевірте роботу нових команд

Головна ідея

Не ви запускаєте команди.

Система запускає їх за вас.

Автоматизація
Звільнює час

Ефективність
Зменшує помилки

Продуктивність
Масштабування

Bash + .bashrc + Cron = Потужна автоматизація