

Пример архитектуры — Bash

Юрий Литвинов
yurii.litvinov@gmail.com

23.01.2019г

Критерии оценивания

- ▶ Нелинейная шкала:
 - ▶ 70% заданий – 2 балла
 - ▶ 75% заданий – 4 балла
 - ▶ 80% заданий – 6 баллов
 - ▶ 90% заданий – 9 баллов
 - ▶ 100% заданий – 10 баллов
- ▶ Задача про CLI оценивается в 20 баллов, остальные (пока что) в 10
- ▶ Оценки за работу на паре с небольшим весом (ближайшая такая пара через две недели)
- ▶ В итоговой оценке практика учитывается с весом 0.4, экзамен — 0.6
- ▶ Округление арифметическое
- ▶ Две оценки — в конце этого и следующего модулей

Enterprise Fizz-Buzz

Задача:

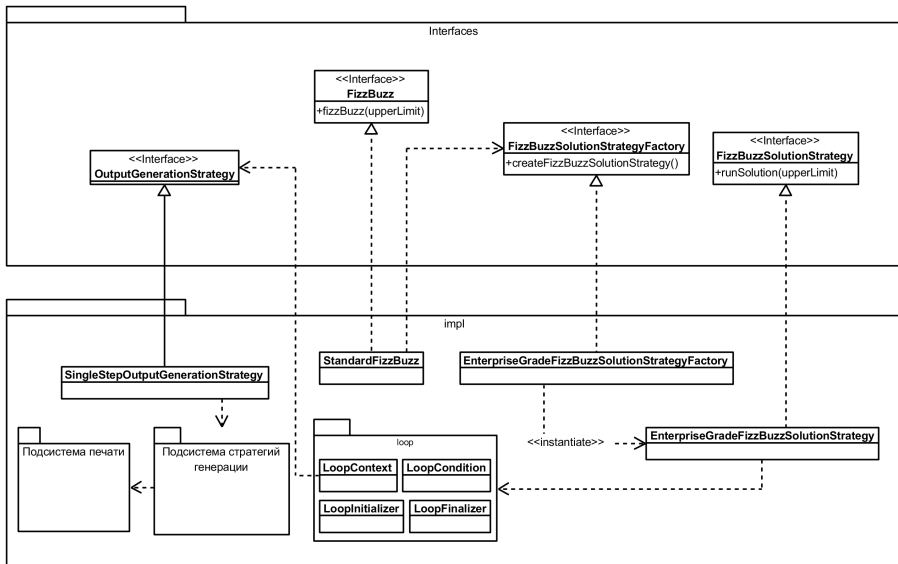
Для чисел от 1 до 100:

- ▶ если число делится на 3, вывести “Fizz”
- ▶ если число делится на 5, вывести “Buzz”
- ▶ если число делится и на 3, и на 5, вывести “FizzBuzz”
- ▶ во всех остальных случаях вывести само число

Решение:

<https://github.com/EnterpriseQualityCoding/FizzBuzzEnterpriseEdition>

Структура системы



Хорошие идеи

- ▶ Separation of Concerns
- ▶ Dependency Inversion
- ▶ Dependency Injection
 - ▶ Spring Framework
- ▶ Паттерны “Фабрика”, “Стратегия”, “Посетитель”, “Адаптер”, что-то вроде паттернов “Спецификация” и “Цепочка ответственности”

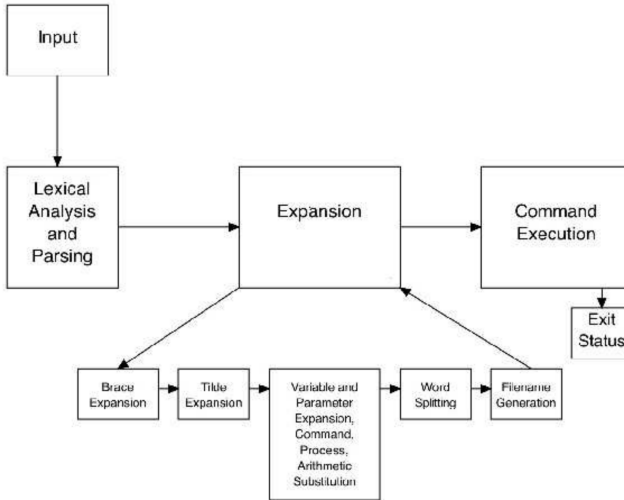
Плохие идеи

- ▶ Не выполняется принцип Keep It Simple Stupid
 - ▶ Неправильно говорить “строк кода написано”, правильно — “строк кода израсходовано”
- ▶ “Синтаксическое” разделение на пакеты, а не “семантическое”
 - ▶ Отсутствие модульности, антипаттерн “Big Ball of Mud”
- ▶ Хардкод основных параметров вычисления
- ▶ Нет юнит-тестов, только интеграционные; нет логирования
- ▶ 1663 строки кода и всего 40 строк комментариев
 - ▶ Отсутствие архитектурного описания

Bash

- ▶ Примерно 70K строк кода
- ▶ Исходный автор — Brian Fox, maintainer — Chet Ramey
- ▶ Первый релиз — 1989
- ▶ Написан на C
- ▶ Архитектурное описание — глава в *The Architecture of Open Source Applications*, написанная Chet Ramey

Архитектура Bash



Основные структуры данных

```
typedef struct word_desc {
    char *word; /* Zero terminated string. */
    int flags; /* Flags associated with this word. */
} WORD_DESC;
```

```
typedef struct word_list {
    struct word_list *next;
    WORD_DESC *word;
} WORD_LIST;
```

Ввод с консоли

- ▶ Библиотека Readline
 - ▶ независимая библиотека, но пишется в основном для Bash
- ▶ Цикл read/dispatch/execute/redisplay
- ▶ Dispatch table (или Keymap)
- ▶ Буфер редактирования, хитрый механизм расчёта действий для отображения
- ▶ Хранит все данные как 8-битные символы, но знает про Unicode

Синтаксический разбор

- ▶ Зависимый от контекста лексический анализ
`for for in for; do for=for; done; echo $for`
- ▶ Использует lex + bison
- ▶ Подстановка alias-ов выполняется лексером
- ▶ Сохранение и восстановление состояния парсера

Подстановки

`${parameter:-word}`

раскрывается в *parameter*, если он установлен, и в *word*, если нет

`pre{one,two,three}post`

раскрывается в

`preonepost pretwopost prethreepost`

Ещё бывает подстановка тильды и арифметическая подстановка, сопоставление шаблона

Исполнение команд

- ▶ Встроенные и внешние команды, обрабатываются единообразно
- ▶ Перенаправление ввода-вывода, отмена перенаправления
- ▶ Принимают набор слов
 - ▶ Иногда обрабатывают по-особому, например, присваивание в *export*
- ▶ Присваивание — тоже команда, но особая
- ▶ Перед запуском внешней команды — поиск в PATH, кеширование результатов
- ▶ Job control, foreground и background

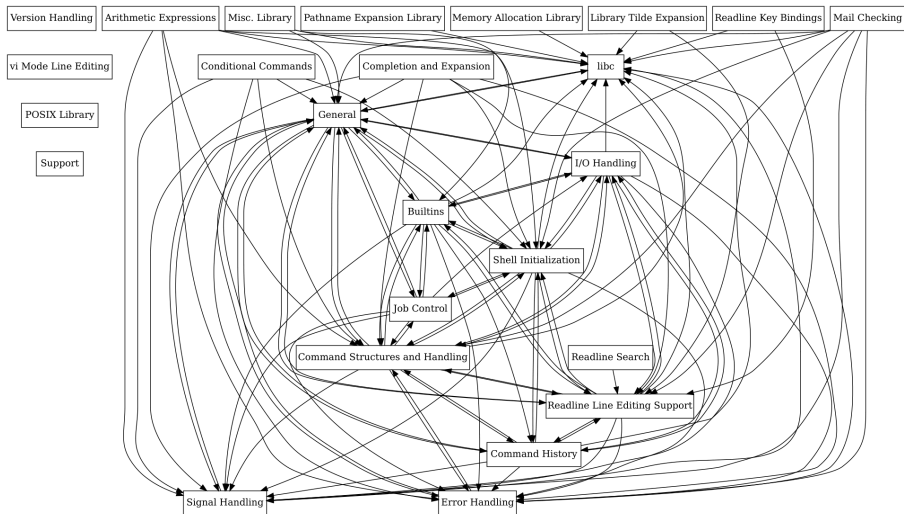
Lessons Learned

- ▶ Комментарии к коммитам со ссылками на багрепорты с шагами воспроизведения
- ▶ Хороший набор тестов, в Bash их тысячи
- ▶ Стандарты, как внешние на функциональность шелла, так и на код
- ▶ Пользовательская документация
- ▶ Переиспользование

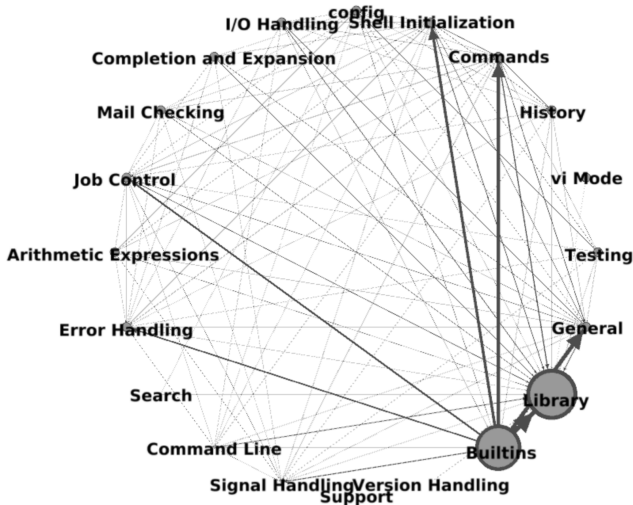
Архитектура Bash, на самом деле

- ▶ J. Garcia et al., *Obtaining Ground-Truth Software Architectures*
- ▶ 1 аспирант, 80 часов работы
- ▶ Верификация от Chet Ramey
- ▶ 70K строк кода, 200 файлов, 25 компонент
 - ▶ 16 — ядро, 9 — утилиты
- ▶ Структура папок почти не соответствует выделенным компонентам

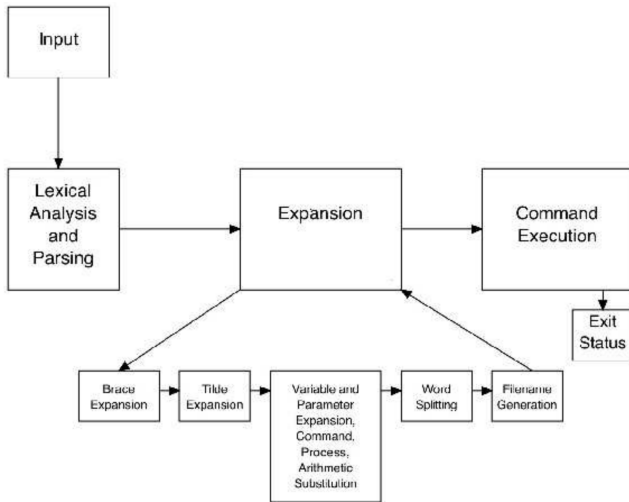
Архитектура Bash, на самом деле



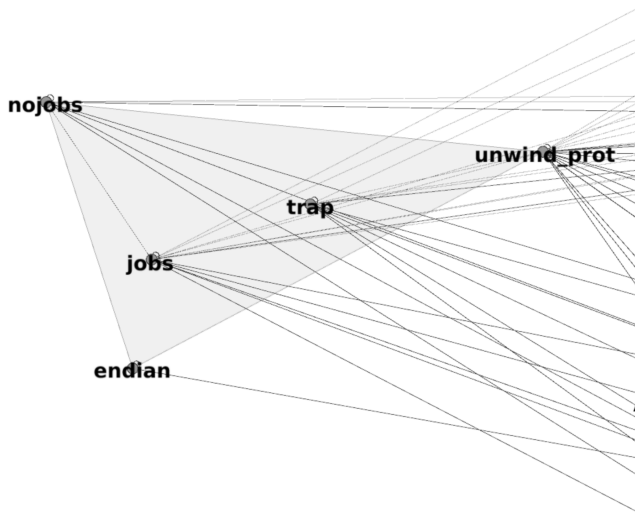
Результаты анализа кода



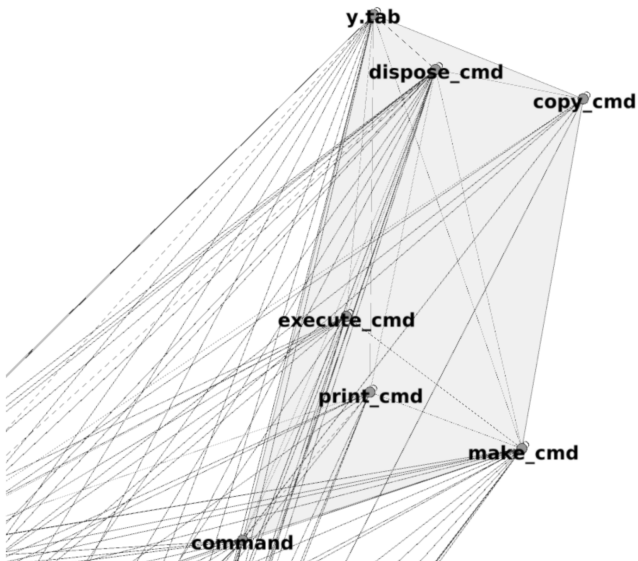
Сравним с исходной



Job Control



Commands



Grep

Следующая задача

Реализовать команду `grep`,

- ▶ поддерживающую ключи
 - ▶ `-i` (нечувствительность к регистру)
 - ▶ `-w` (поиск только слов целиком)
 - ▶ `-A n` (распечатать `n` строк после строки с совпадением)
- ▶ поддерживающую регулярные выражения в строке поиска
- ▶ использующую одну из библиотек для разбора аргументов командной строки

Примеры

- > grep plugin build.gradle
 apply plugin: 'java'
 apply plugin: 'idea'
- > cat build.gradle | grep plugin
 apply plugin: 'java'
 apply plugin: 'idea'
- > grep -A 2 plugin build.gradle
 apply plugin: 'java'
 apply plugin: 'idea'
 group = 'ru.example'
 version = '1.0'

Замечания

- ▶ Ожидается обоснование выбора библиотеки для работы с аргументами
 - ▶ Какие библиотеки были рассмотрены
 - ▶ Почему выбрана именно та, что выбрана
 - ▶ кратко описать текстом
- ▶ Сдавать как новый пуллреквест из новой ветки на базе предыдущей