

Лекция 15: Примеры архитектур

Юрий Литвинов
y.litvinov@spbu.ru

15.12.2022

Enterprise Fizz-Buzz

Задача:

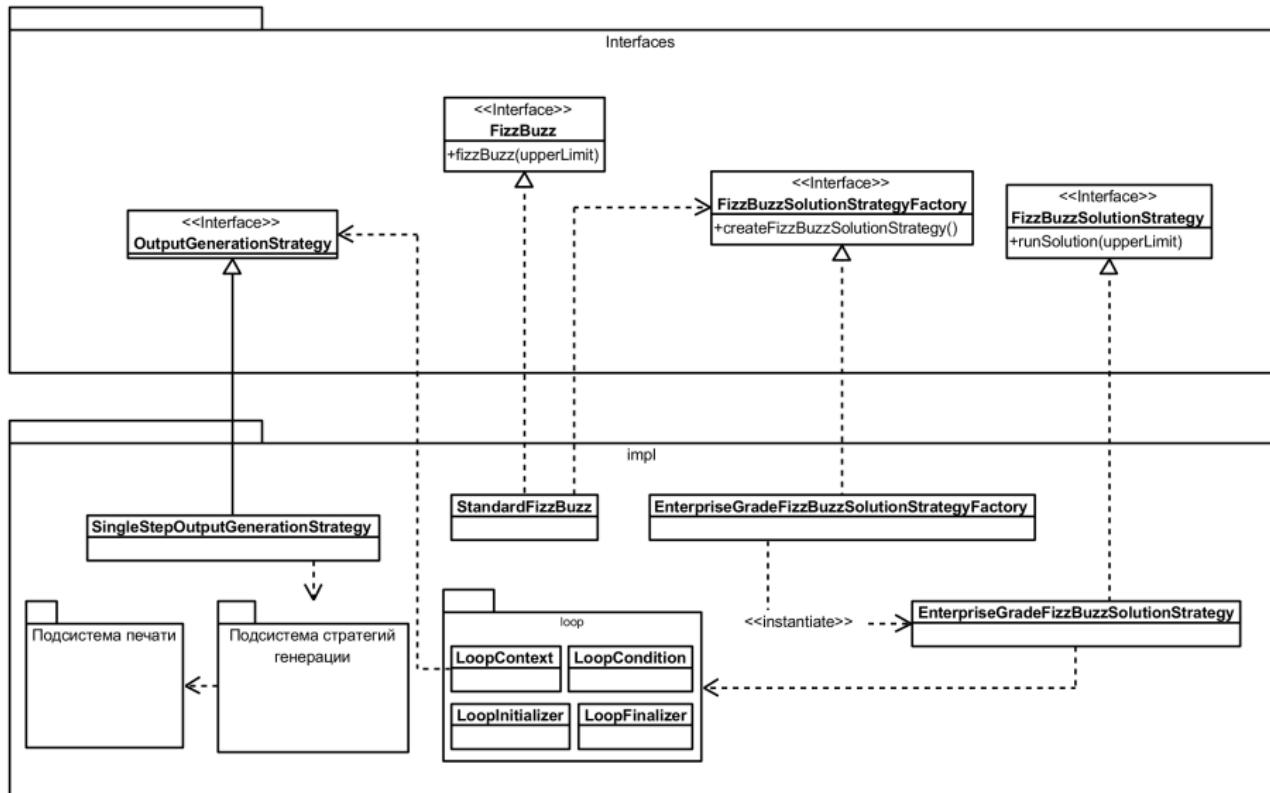
Для чисел от 1 до 100:

- ▶ 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”

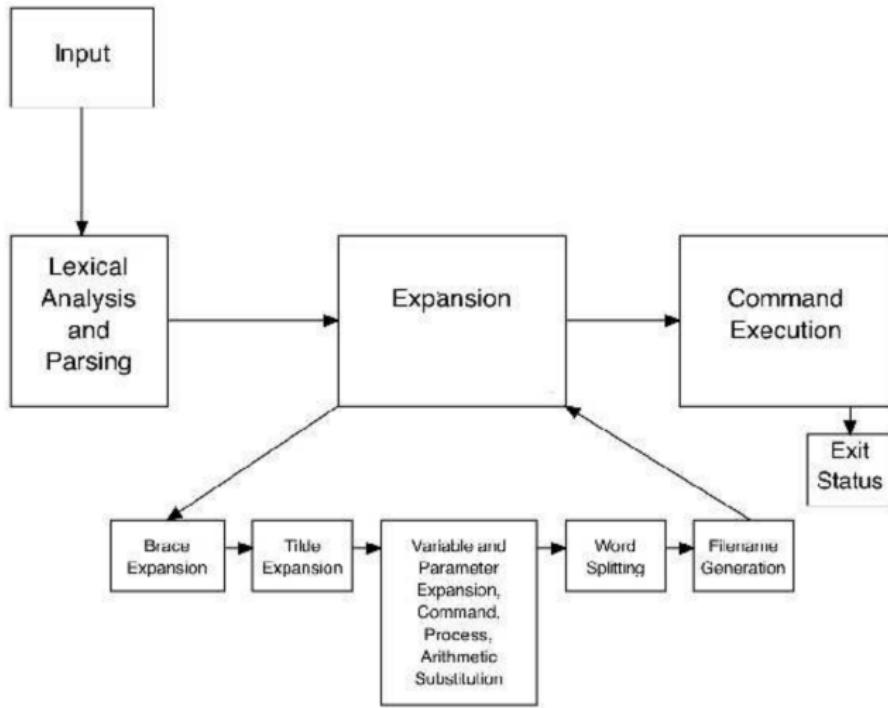
Плохие идеи

- ▶ 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 70К строк кода
- ▶ Исходный автор — Brian Fox, maintainer — Chet Ramey
- ▶ Первый релиз — 1989
- ▶ Написан на С
- ▶ 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

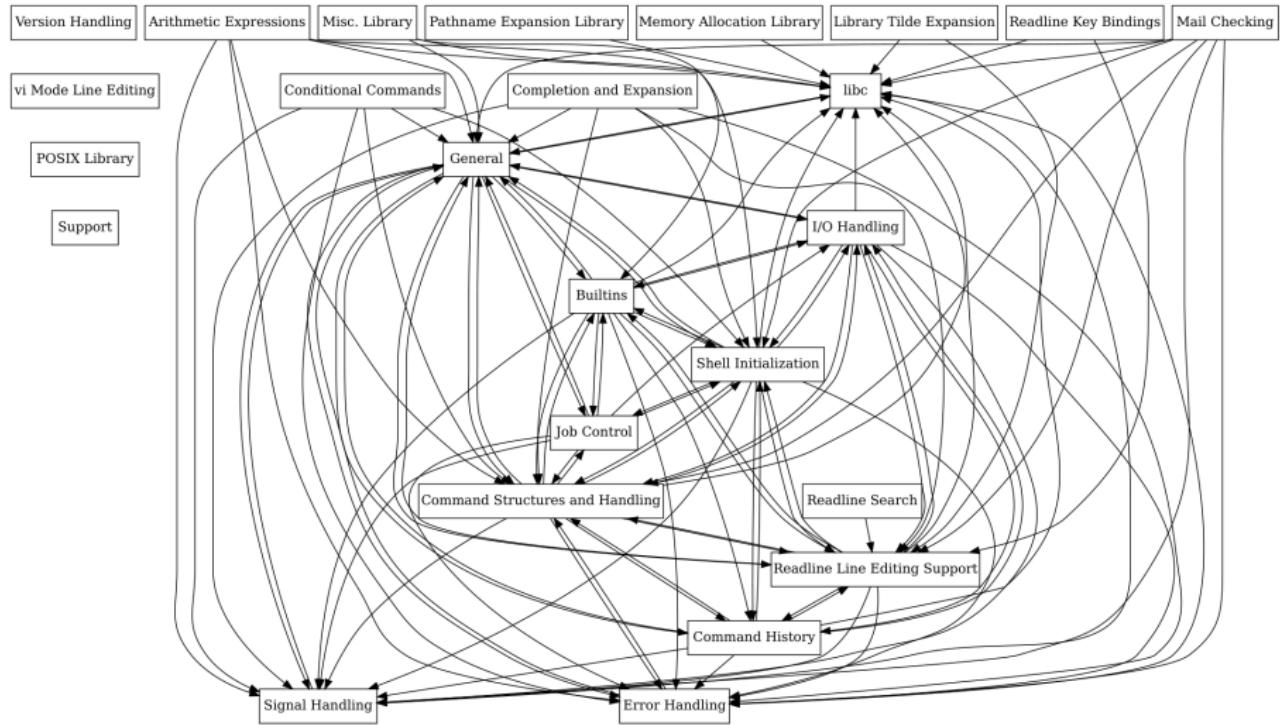
Lessons Learned

- ▶ 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

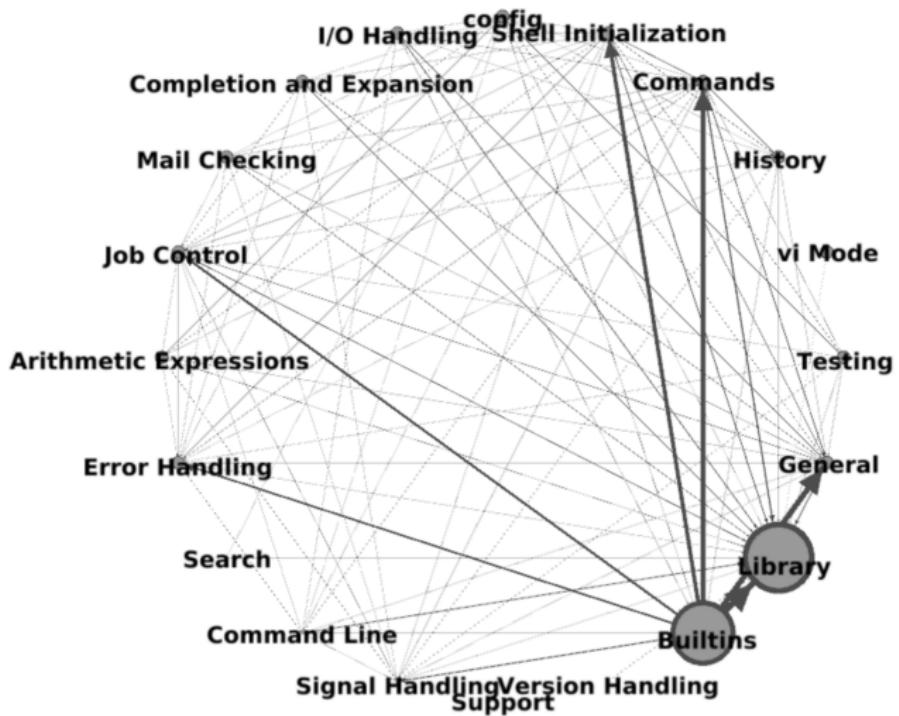
Архитектура Bash, на самом деле

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

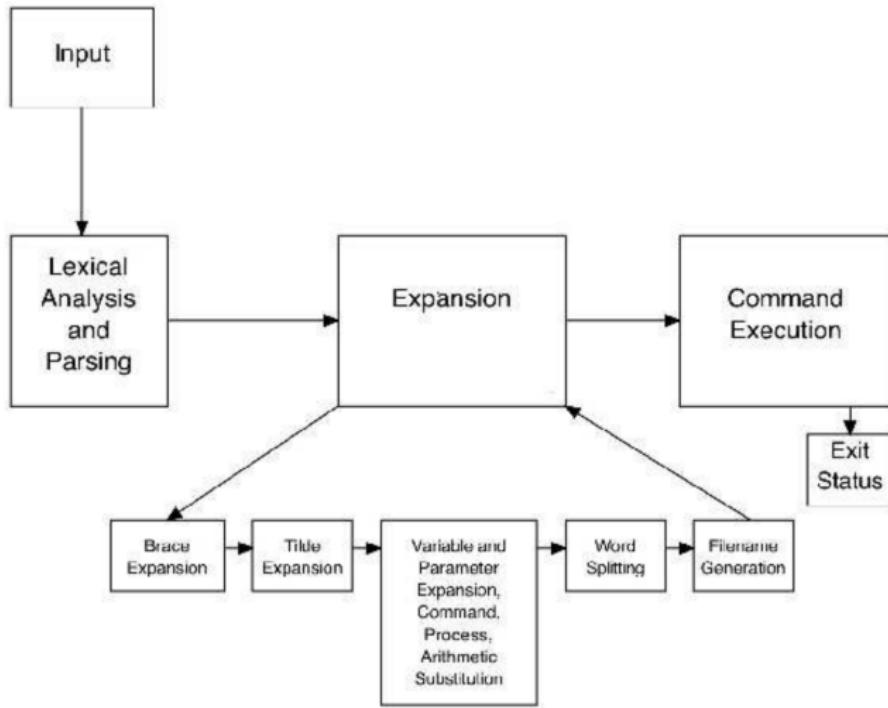
Архитектура Bash, на самом деле



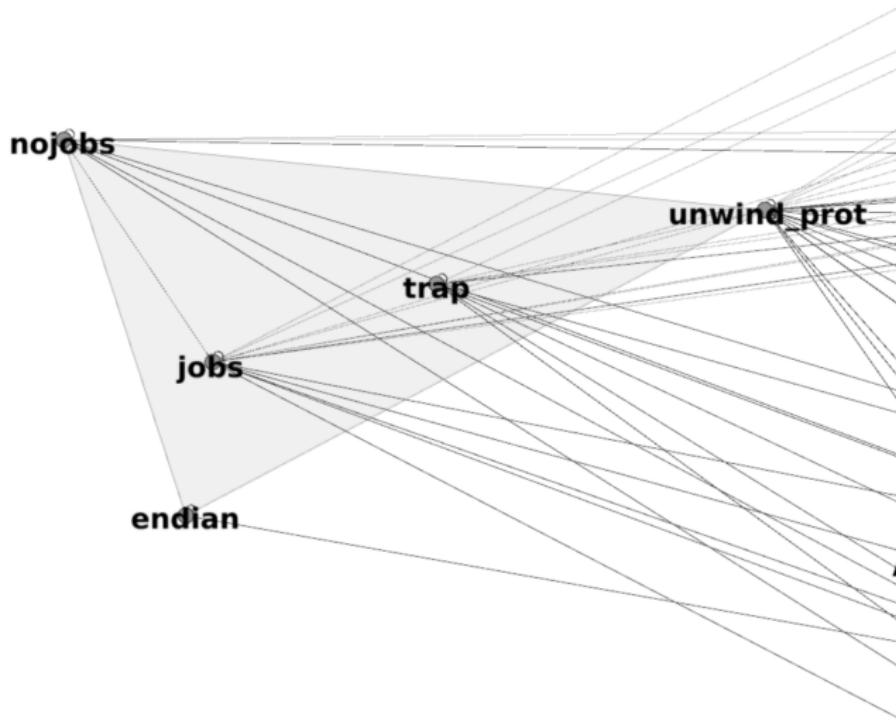
Результаты анализа кода



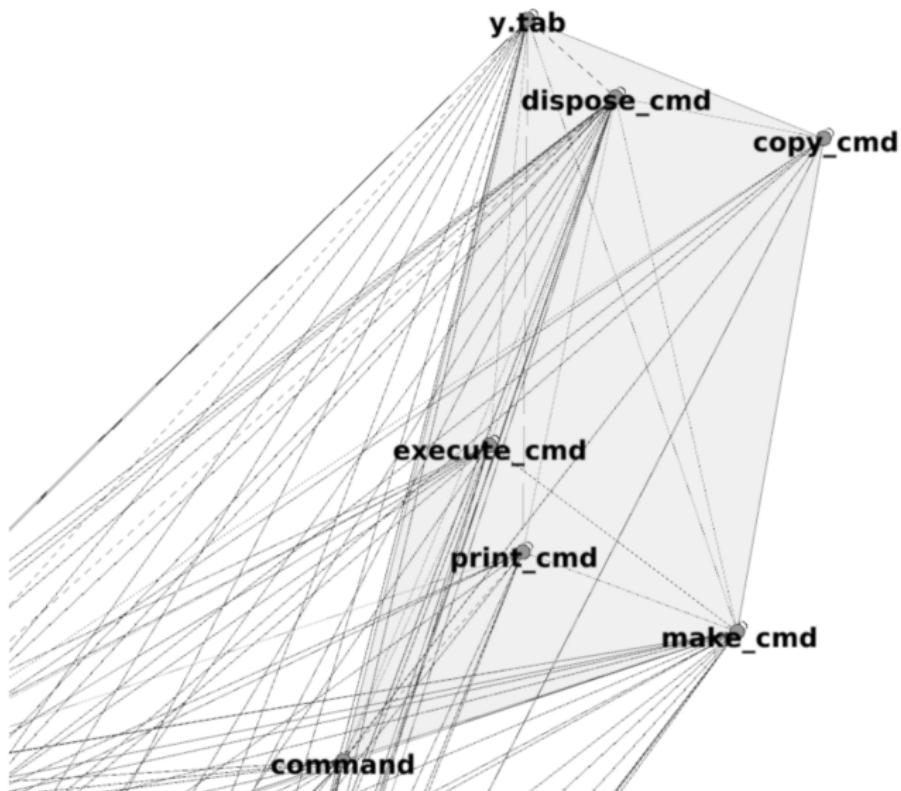
Сравним с исходной



Job Control



Commands



Git¹

- ▶ Распределённая VCS
- ▶ Linus Torvalds, 2005 год, драма с BitKeeper
- ▶ Architectural drivers
 - ▶ Распределённая разработка с тысячей коммитеров
 - ▶ Защита от порчи исходников
 - ▶ Возможность отменить мердж, смерджиться вручную
 - ▶ Высокая скорость работы

¹По гл. 10 <https://git-scm.com/book> и <http://aosabook.org/en/git.html>

Внутреннее устройство Git

Структура папки .git:

- ▶ HEAD
- ▶ index
- ▶ config
- ▶ description
- ▶ hooks/
- ▶ info/
- ▶ objects/
- ▶ refs/
- ▶ ...

Объекты

Git внутри — хеш-таблица, отображающая SHA-1-хеш файла в содержимое файла. Пример:

```
$ git init test
```

```
Initialized empty Git repository in /tmp/test/.git/
```

```
$ cd test
```

```
$ find .git/objects
```

```
.git/objects
```

```
.git/objects/info
```

```
.git/objects/pack
```

```
$ echo 'test content' | git hash-object -w --stdin
```

```
d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

```
$ find .git/objects -type f
```

```
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

Объекты (2)

Как получить сохранённый объект:

```
$ git cat-file -p d670460b4b4aece5915caf5c68d12f560a9fe3e4  
test content
```

Версионный контроль:

```
$ echo 'version 1' > test.txt  
$ git hash-object -w test.txt  
83baae61804e65cc73a7201a7252750c76066a30  
$ echo 'version 2' > test.txt  
$ git hash-object -w test.txt  
1f7a7a472abf3dd9643fd615f6da379c4acb3e3a  
$ find .git/objects -type f  
.git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a  
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30  
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

Объекты (3)

Переключение между версиями файла:

```
$ git cat-file -p 83baae61804e65cc73a7201a7252750c76066a30 \
  > test.txt
```

```
$ cat test.txt
```

version 1

```
$ git cat-file -p 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a \
  > test.txt
```

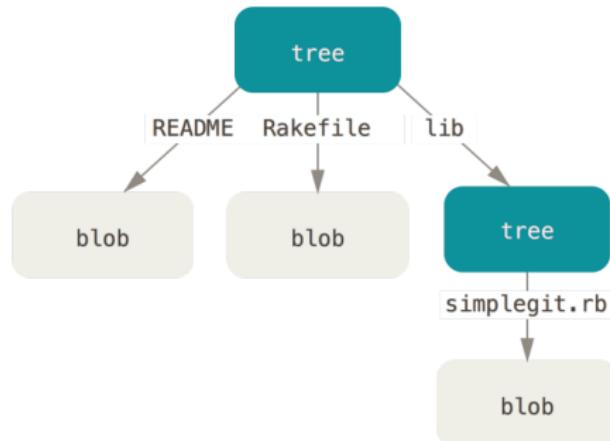
```
$ cat test.txt
```

version 2

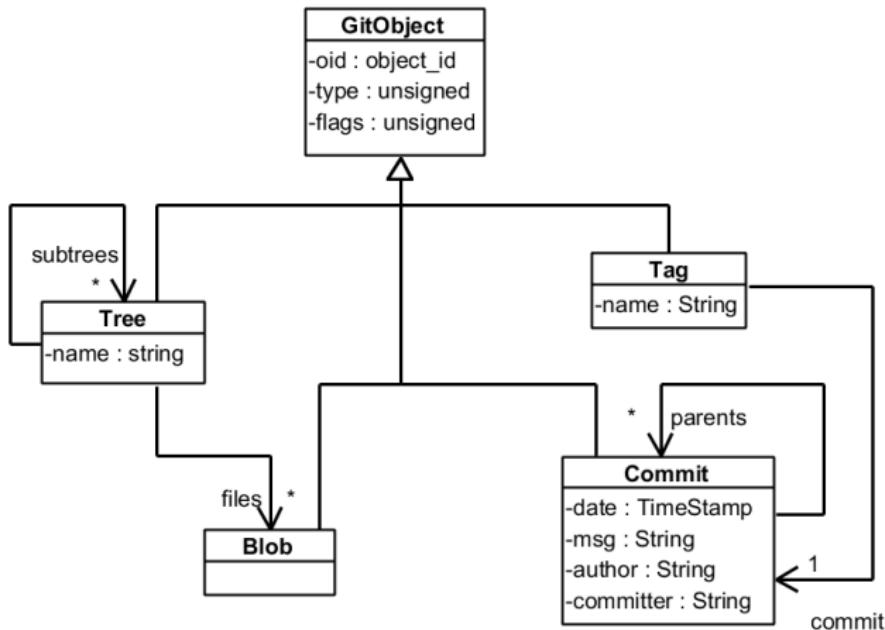
Деревья

blob (то, что мы видели раньше) хранит только содержимое файла, не хранит даже его имя. Решение проблемы — tree:

```
$ git cat-file -p master^{tree}
100644 blob a906cb2a4a904a152e80877d4088654daad0c859 README
100644 blob 8f94139338f9404f26296befa88755fc2598c289 Rakefile
040000 tree 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0 lib
```



Какие ещё виды объектов бывают



Коммиты

tree-объекты могут хранить структуру файлов (как inode в файловой системе), но не хранят метаинформацию типа автора файла и даты создания. Это хранится в commit-объектах:

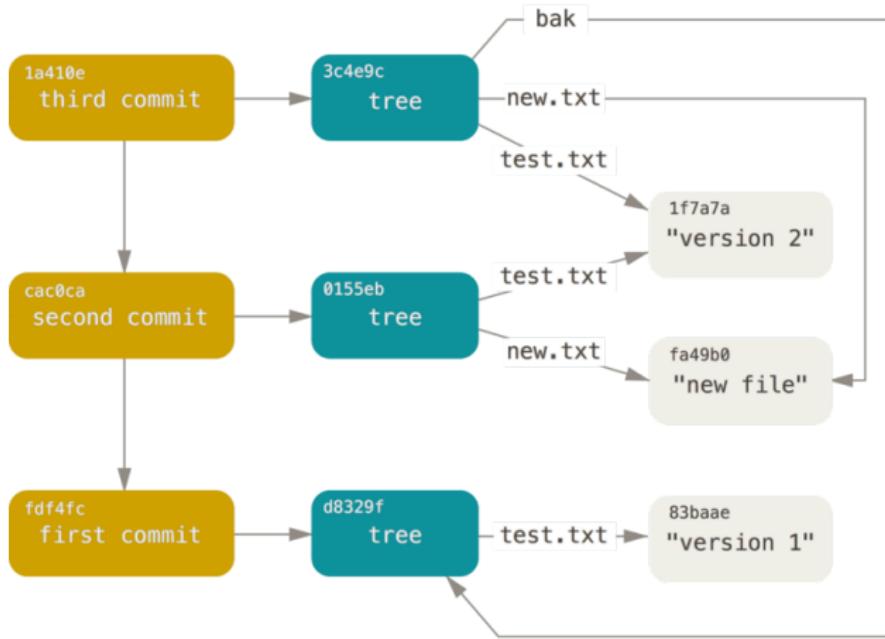
```
$ echo 'first commit' | git commit-tree d8329f  
fdf4fc3344e67ab068f836878b6c4951e3b15f3d
```

```
$ git cat-file -p fdf4fc3  
tree d8329fc1cc938780ffdd9f94e0d364e0ea74f579  
author Scott Chacon <schacon@gmail.com> 1243040974 -0700  
committer Scott Chacon <schacon@gmail.com> 1243040974 -0700
```

first commit

Ещё коммит хранит список коммитов-родителей

Коммиты, как это выглядит



Ссылки

Теперь вся информация хранится на диске, но чтобы ей воспользоваться, нужно помнить SHA-1 хеши. На помощь приходят reference-ы.

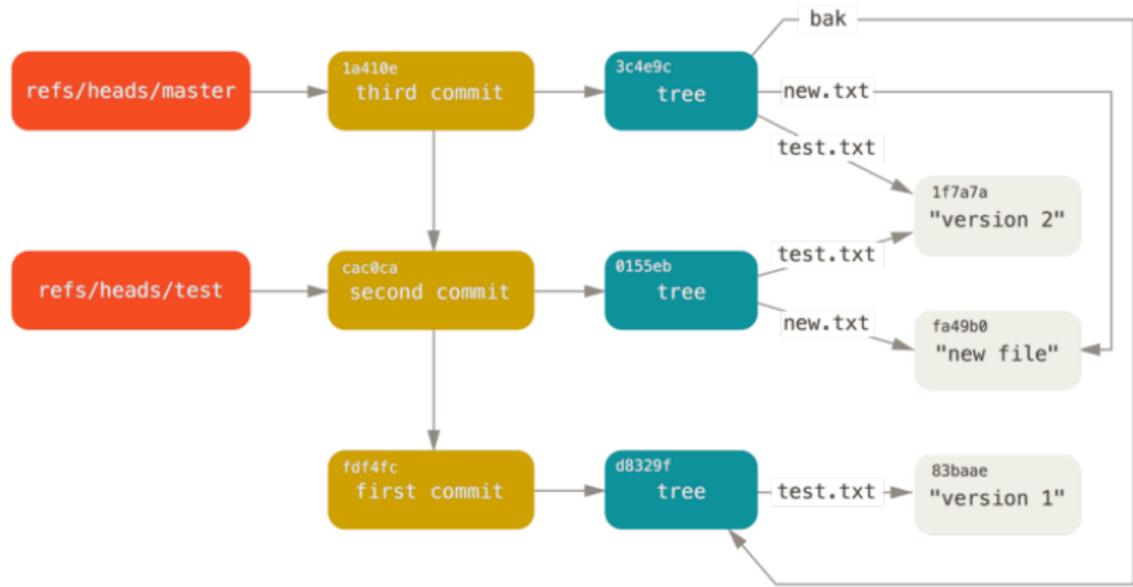
- ▶ .git/refs
- ▶ .git/refs/heads
- ▶ .git/refs/tags

```
$ echo "1a410efbd13591db07496601ebc7a059dd55cfe9" \  
 > .git/refs/heads/master
```

```
$ git log --pretty=oneline master  
1a410efbd13591db07496601ebc7a059dd55cfe9 third commit  
cac0cab538b970a37ea1e769ccbde608743bc96d second commit  
fdf4fc3344e67ab068f836878b6c4951e3b15f3d first commit
```

- ▶ Команда git update-ref

Ссылки, как это выглядит



HEAD

Теперь не надо помнить хеши, но как переключаться между ветками?

Текущая ветка хранится в HEAD. HEAD — символьическая ссылка, то есть ссылка на другую ссылку.

```
$ cat .git/HEAD  
ref: refs/heads/master
```

```
$ git symbolic-ref HEAD refs/heads/test  
$ cat .git/HEAD  
ref: refs/heads/test
```

Тэги

Последний из объектов в Git — tag. Это просто указатель на коммит.

- ▶ Легковесный тэг:

```
git update-ref refs/tags/v1.0 cac0cab538b970a37ea1e769ccbde608743bc96d
```

Или просто git tag

- ▶ Аннотированный тэг:

```
$ git tag -a v1.1 1a410efbd13591db07496601ebc7a059dd55cfe9 -m 'test tag'
```

```
$ git cat-file -p 9585191f37f7b0fb9444f35a9bf50de191beadc2
```

```
object 1a410efbd13591db07496601ebc7a059dd55cfe9
```

```
type commit
```

```
tag v1.1
```

```
tagger Scott Chacon <schacon@gmail.com> Sat May 23 16:48:58 2009 -0700
```

```
test tag
```

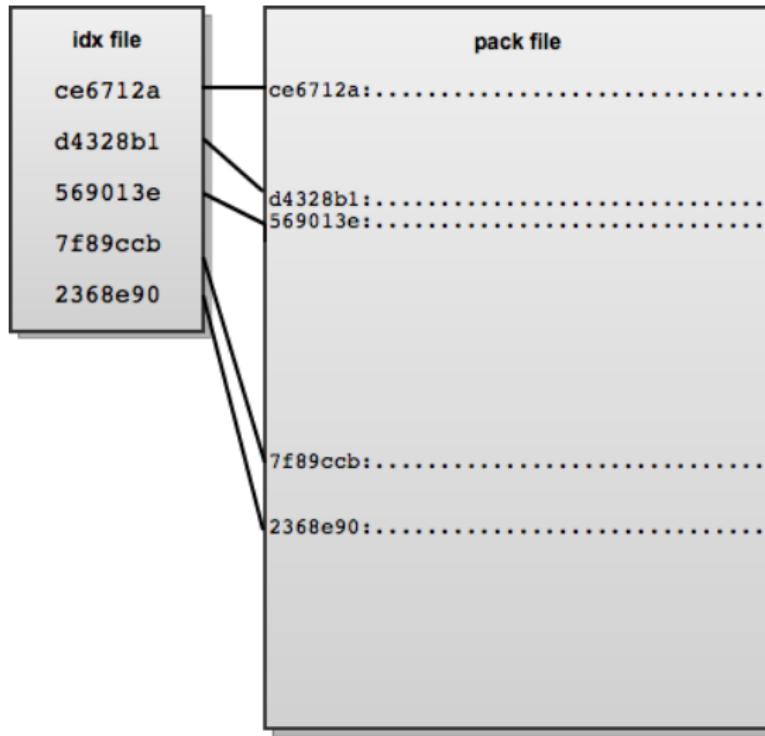
Packfiles

Пока что получалось, что все версии всех файлов в Git хранятся целиком, как они есть. Все они всегда сжимаются zlib, но в целом, если создать репозиторий, добавлять туда файлы, коммитить и т.д., все версии всех файлов будут в нём целиком. На помощь приходят .pack-файлы:

```
$ git gc
Counting objects: 18, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (18/18), done.
Total 18 (delta 3), reused 0 (delta 0)
```

```
$ find .git/objects -type f
.git/objects/bd/9dbf5aae1a3862dd1526723246b20206e5fc37
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects/info/packs
.git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.idx
.git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.pack
```

Как оно устроено



Pack-файлы, подробности

- ▶ Упаковка происходит, когда:
 - ▶ Выполняется git push
 - ▶ Слишком много «свободных» объектов (порядка 7000)
 - ▶ Вручную вызвана git gc
- ▶ Используется дельта-компрессия
 - ▶ Последняя версия хранится целиком, дельты «идут назад»
- ▶ Можно заглянуть внутрь, git verify-pack
- ▶ Git может хитро перепаковывать pack-файлы

Reflog и восстановление коммитов

```
$ git reflog  
1a410ef HEAD@{0}: reset: moving to 1a410ef  
ab1afef HEAD@{1}: commit: modified repo.rb a bit  
484a592 HEAD@{2}: commit: added repo.rb
```

```
$ git log -g  
commit 1a410efbd13591db07496601ebc7a059dd55cfe9  
Reflog: HEAD@{0} (Scott Chacon <schacon@gmail.com>)  
Reflog message: updating HEAD  
Author: Scott Chacon <schacon@gmail.com>  
Date: Fri May 22 18:22:37 2009 -0700
```

third commit

```
$ git branch recover-branch ab1afef
```

Как более капитально прострелить себе ногу

И что делать

```
$ git branch -D recover-branch  
$ rm -Rf .git/logs/
```

```
$ git fsck --full
```

Checking object directories: 100% (256/256), done.

Checking objects: 100% (18/18), done.

dangling blob d670460b4b4aece5915caf5c68d12f560a9fe3e4

dangling commit ab1afef80fac8e34258ff41fc1b867c702daa24b

dangling tree aea790b9a58f6cf6f2804eeac9f0abbe9631e4c9

dangling blob 7108f7ecb345ee9d0084193f147cdad4d2998293

Git не удалит даже «висячие» объекты несколько месяцев, если его явно не попросить.

Lessons Learned

- ▶ Команды реализовывались как набор шелл-скриптов
 - ▶ Не портировать под Windows
 - ▶ Сложно интегрировать с IDE
 - ▶ Замедлило внедрение git-a
- ▶ Большой набор команд (включая plumbing) делает Git тяжёлым для изучения и усложняет сообщения об ошибках

Battle for Wesnoth²

- ▶ Пошаговая стратегия
- ▶ Порядка 200000 строк кода на C++
- ▶ 4 миллиона скачиваний
- ▶ 9/10 на Steam
- ▶ 2003 год



© <https://www.wesnoth.org/>

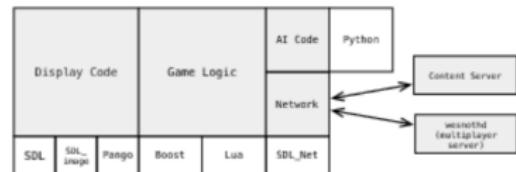
²По <http://aosabook.org>

Architectural Drivers

- ▶ Доступность для новых разработчиков и авторов контента
- ▶ В ущерб технической красоте
- ▶ Не nice to have, а условие выживания проекта в контексте широкого open-source сообщества из людей без каких-либо обязательств и разного технического уровня

Высокоуровневая архитектура

- ▶ Wesnoth Markup Language (WML)
- ▶ Минимизация зависимостей от сторонних библиотек
 - ▶ SDL (Simple Directmedia Layer) для видео и ввода/вывода
 - ▶ Простота использования и кроссплатформенность
 - ▶ Boost, Pango, zlib, Python, Lua, GNU gettext



Основные компоненты

- ▶ Парсер и препроцессор WML
- ▶ Базовый ввод-вывод — видео, звук, сеть
- ▶ GUI — виджеты
- ▶ Display module — игровая доска, юниты, анимация и т.д.
- ▶ ИИ
- ▶ Поиск пути (плюс утилиты для работы с гексагональной доской)
- ▶ Генератор карт
- ▶ Специализированные модули
 - ▶ Титульный экран
 - ▶ Storyline module — для проигрывания катсцен
 - ▶ Лобби — для мультиплеера
 - ▶ “Play game” module — управление основным игровым процессом
- ▶ Отдельно — wesnothd и content server

Wesnoth Markup Language

[unit_type]

 id=Elvish Fighter

 name=_ "Elvish Fighter"

 image="units/elves-wood/fighter.png"

 hitpoints=33

 advances_to=Elvish Captain,Elvish Hero

 {LESS_NIMBLE_ELF}

[attack]

 name=sword

 icon=attacks/sword-elven.png

 range=melee

 damage=5

[/attack]

[/unit_type]

Макросы

```
#define GOLD EASY_AMOUNT NORMAL_AMOUNT HARD_AMOUNT
#define EASY
    gold={EASY_AMOUNT}
#endif
#define NORMAL
    gold={NORMAL_AMOUNT}
#endif
#define HARD
    gold={HARD_AMOUNT}
#endif
#endif
...
{GOLD 50 100 200}
```

Модель данных

- ▶ Всё сливается в один гигантский WML-документ
- ▶ Перезагружается при смене опций
- ▶ Всякие хаки на уровне препроцессора, чтобы не грузить вообще всё
- ▶ Классы unit и unit_type (архитектурный стиль Knowledge Layer)
- ▶ Фиксированный набор поддерживаемых движком атрибутов, задаваемых для каждого типа через WML
 - ▶ Нельзя описывать произвольное поведение через WML, хотели сохранить декларативность
- ▶ Класс attack_type
- ▶ Трейты, инвентарь

Мультиплеер

- ▶ Начальное состояние и команды
- ▶ Сервер просто пересыпает команды между клиентами
 - ▶ TCP/IP
- ▶ Replay
- ▶ Никакой защиты от читов
- ▶ Версии клиентов

Lessons Learned

- ▶ 250 тысяч строк на WML
- ▶ Сотни созданных пользователями кампаний
- ▶ 74 тысячи коммитов, 196 контрибуторов
- ▶ Сами разработчики смеются над WML
- ▶ В целом задача обеспечить доступность для модификации очень сложна