

Введение / погружение в Git



План

Методологии разработки

Системы контроля версий

Continuous Integration / Continuous Delivery

Методологии разработки



Agile



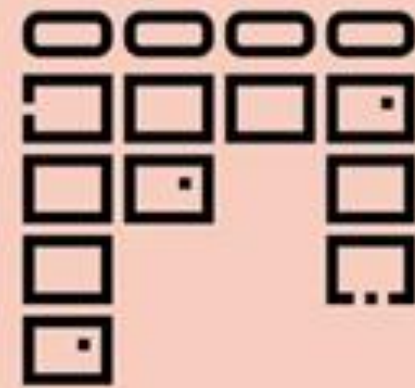
**Creativity
Mindfulness**



Scrum



**Self-knowledge
Cooperation**



Kanban



**Self-management
Work-life balance**



Waterfall

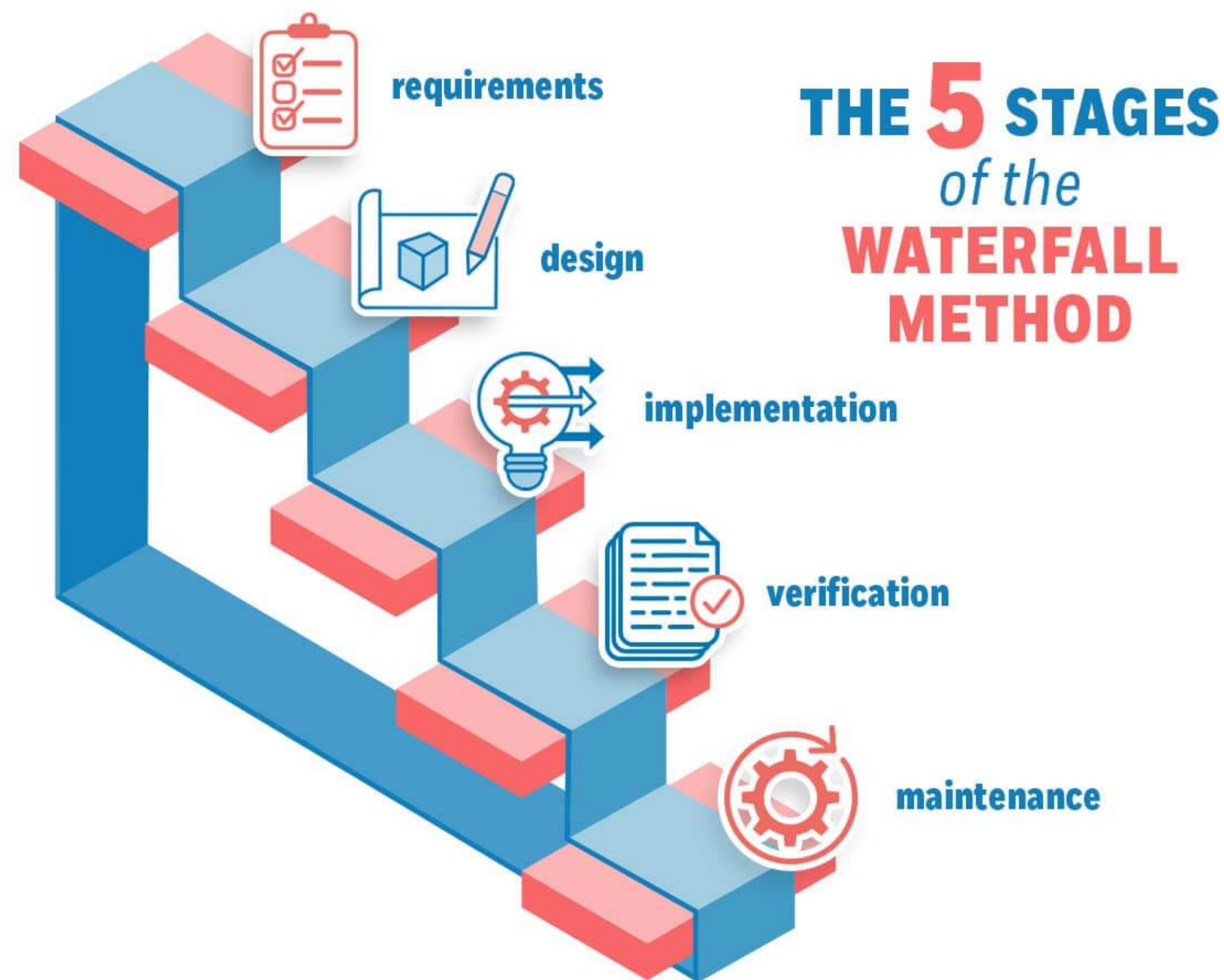


**Planning
Diligence**

Жизненный цикл разработки

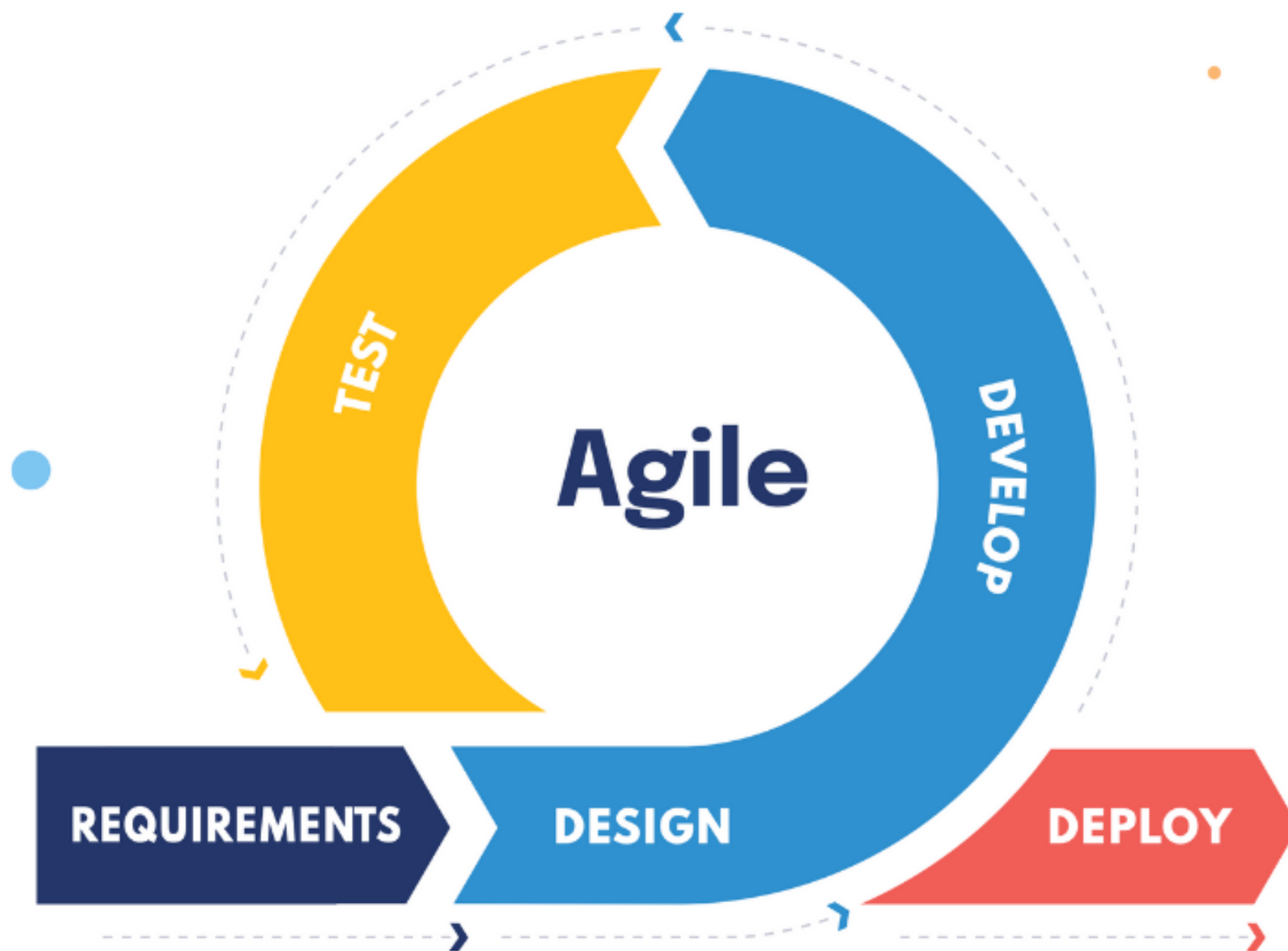


Waterfall (он же водопад, он же каскад)



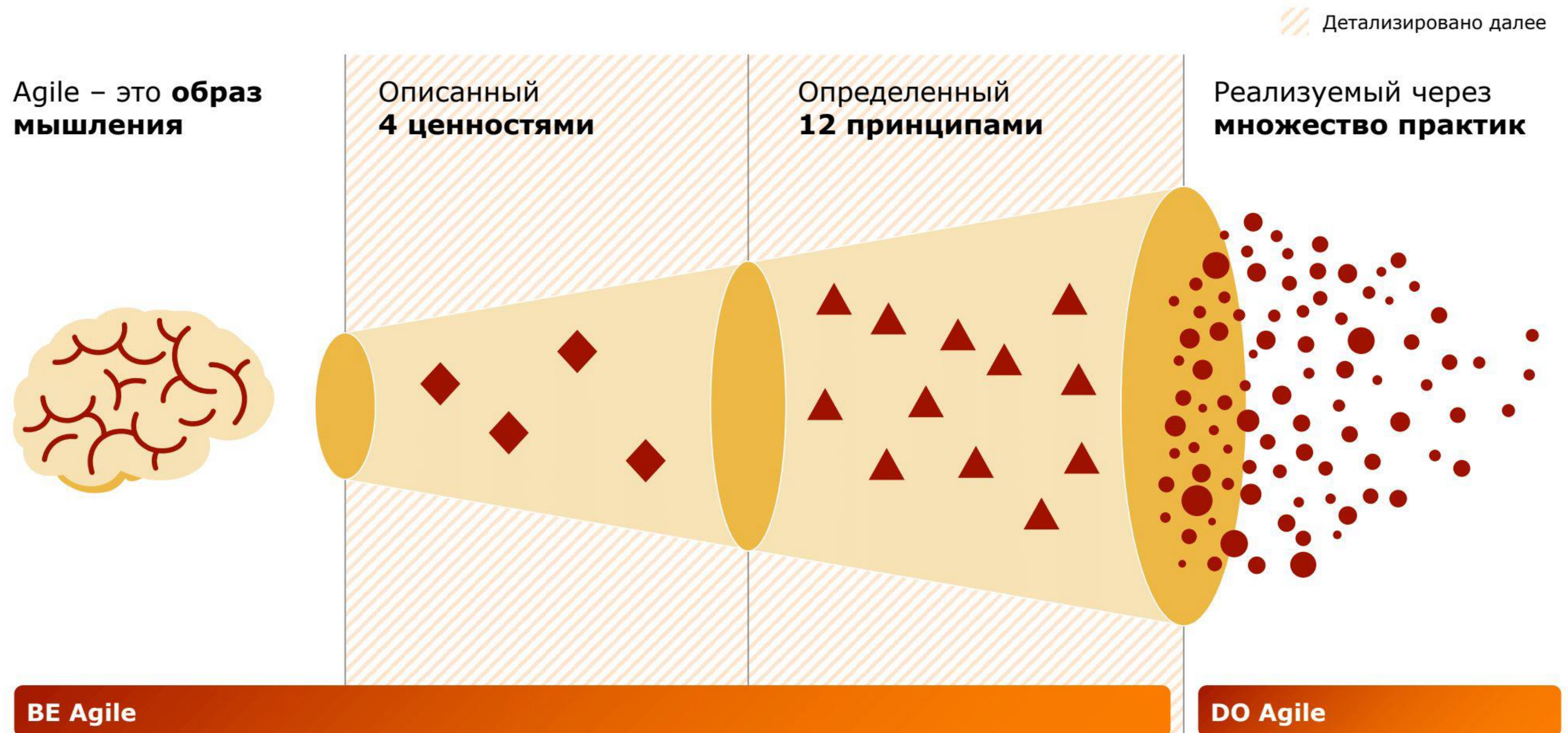
- + Простота управления
- + Ускоренная доставка
- + Хорошо работает для небольших проектов
- + Хорошо документированные процессы
- + Хорошо адаптируется для смены команд
- Лишен гибкости
- Сложность первого этапа
- "Стойкость к изменениям"

Agile (он же гибкий)



- + Ориентация на клиента
- + Поддерживается качество
- + Основан на постепенном прогрессе
- + Команды самоорганизованы
- + Клиенты знают статус - меньше риски
- Плохо подходит для небольших проектов
- Встречи требуют эксперта для принятия решений
- Легко сбиться с пути, если руководство не уверено в результате
- Стоимость внедрения больше













Agile подход



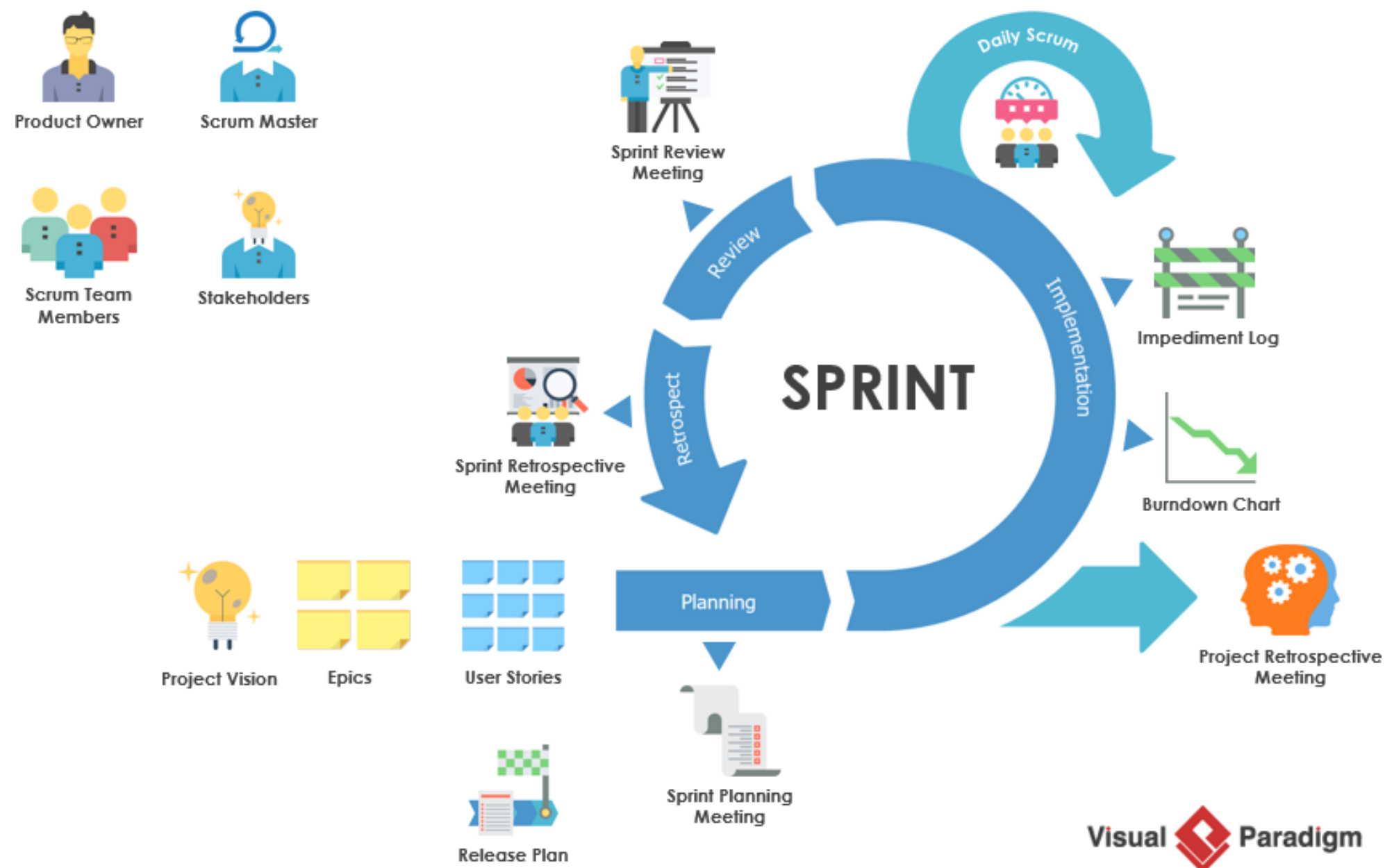
Ценности Agile



Принципы Agile

1	Наивысший приоритет для нас — удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения	
2	Изменение требований приветствуется даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества	
3	Работающий продукт следует выпускать как можно чаще, с периодичностью от нескольких недель до нескольких месяцев	
4	На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе	
5	Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им	
6	Непосредственное общение — наиболее практичный и эффективный способ обмена информацией как с самой командой, так и внутри команды	
7	Работающий продукт — основной показатель прогресса	
8	Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно. Agile помогает наладить такой устойчивый процесс разработки	
9	Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта	
10	Простота — искусство минимизации лишней работы — крайне необходима	
11	Самые лучшие требования, архитектурные и технические решения обеспечивают самоорганизующиеся команды	
12	Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы	

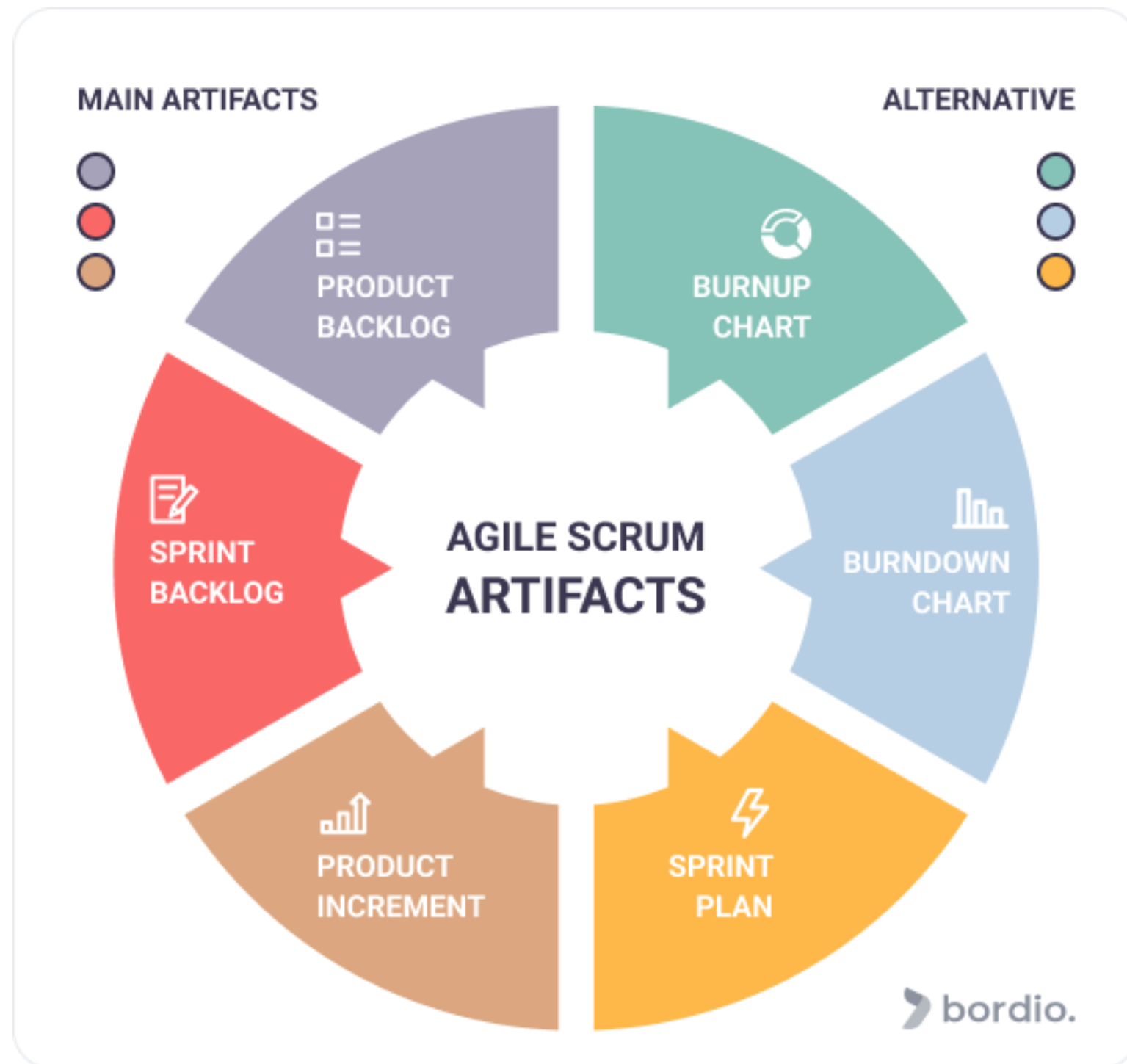
Методология Scrum



Принципы Scrum:

- Структурированный процесс. Работа над проектом разделена на этапы, состоящие из задач, у каждой задачи есть ответственный исполнитель и строгие дедлайны.
- Поэтапная отчетность. Отчет о работе по завершению каждого этапа.
- Подведение итогов командой разработчиков. Scrum-митинг и ретроспектива. Практики Scrum основаны на командном обсуждении задач и их решении.
- Изменения в проекте без потерь. В процессе разработки исполнитель держит связь с заказчиком, чтобы не упустить ни одной детали.

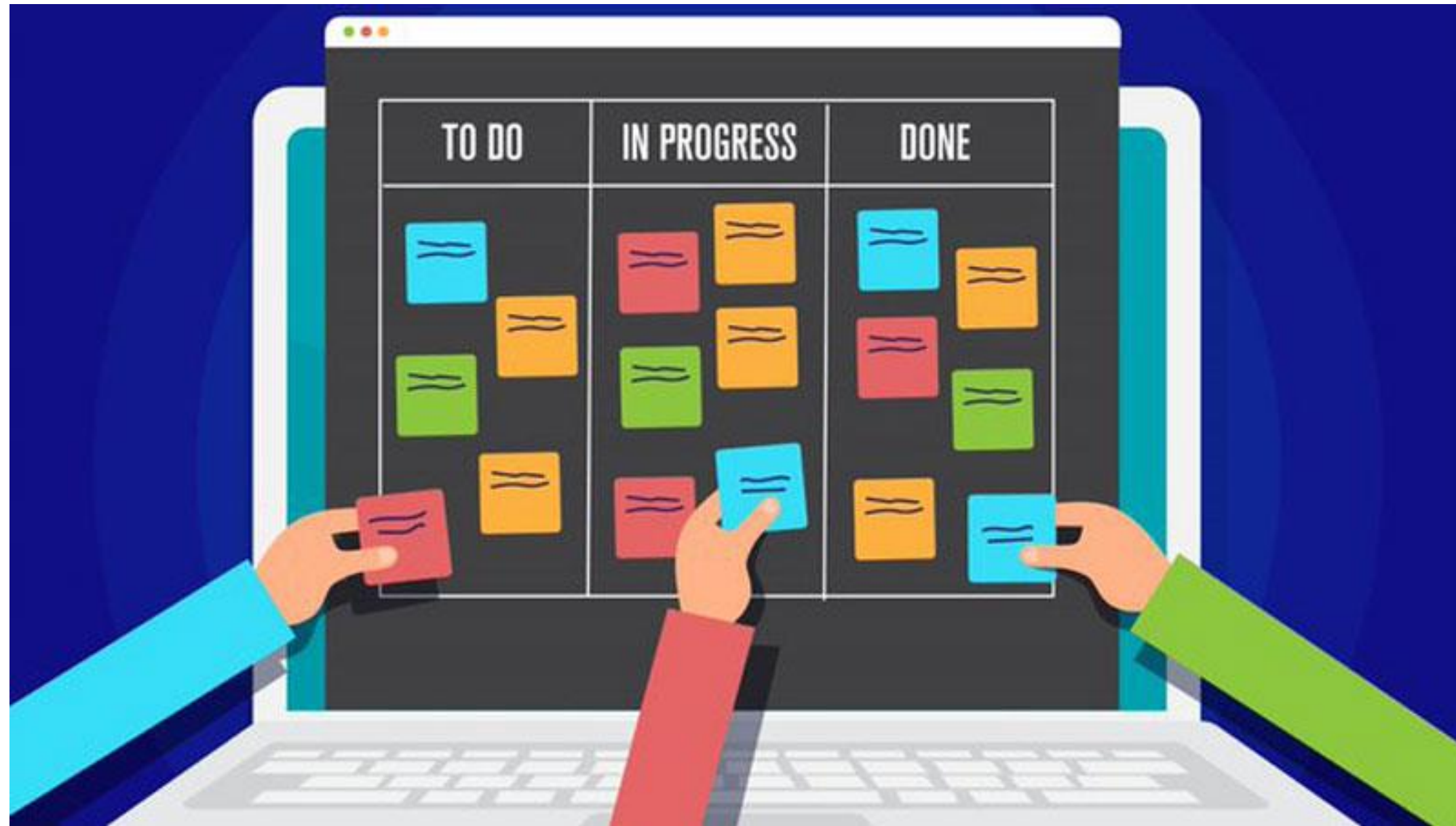
Артефакты Scrum



- Бэклог продукта
- Бэклог спринта
- Цель спринта (план)
- Инкремент
- Burndown чарт спринта
- Burndown чарт релиза

Артефакты обеспечивают прозрачность ключевой информации и создают новые возможности для инспекции и адаптации.

Kanban



Принципы Kanban:

- Использование доски задач. Весь процесс работы отображен на физической доске.
- Не больше 2-х задач в работе. Разработчики не распыляются на лишние моменты, поэтому работают над проектом качественно.
- Команда следит за таймингом: у каждой задачи есть свои дедлайны.
- В процессе разработки команда постоянно совершенствует навыки.

For a smile ;)

Каскадная модель



Agile



Канбан



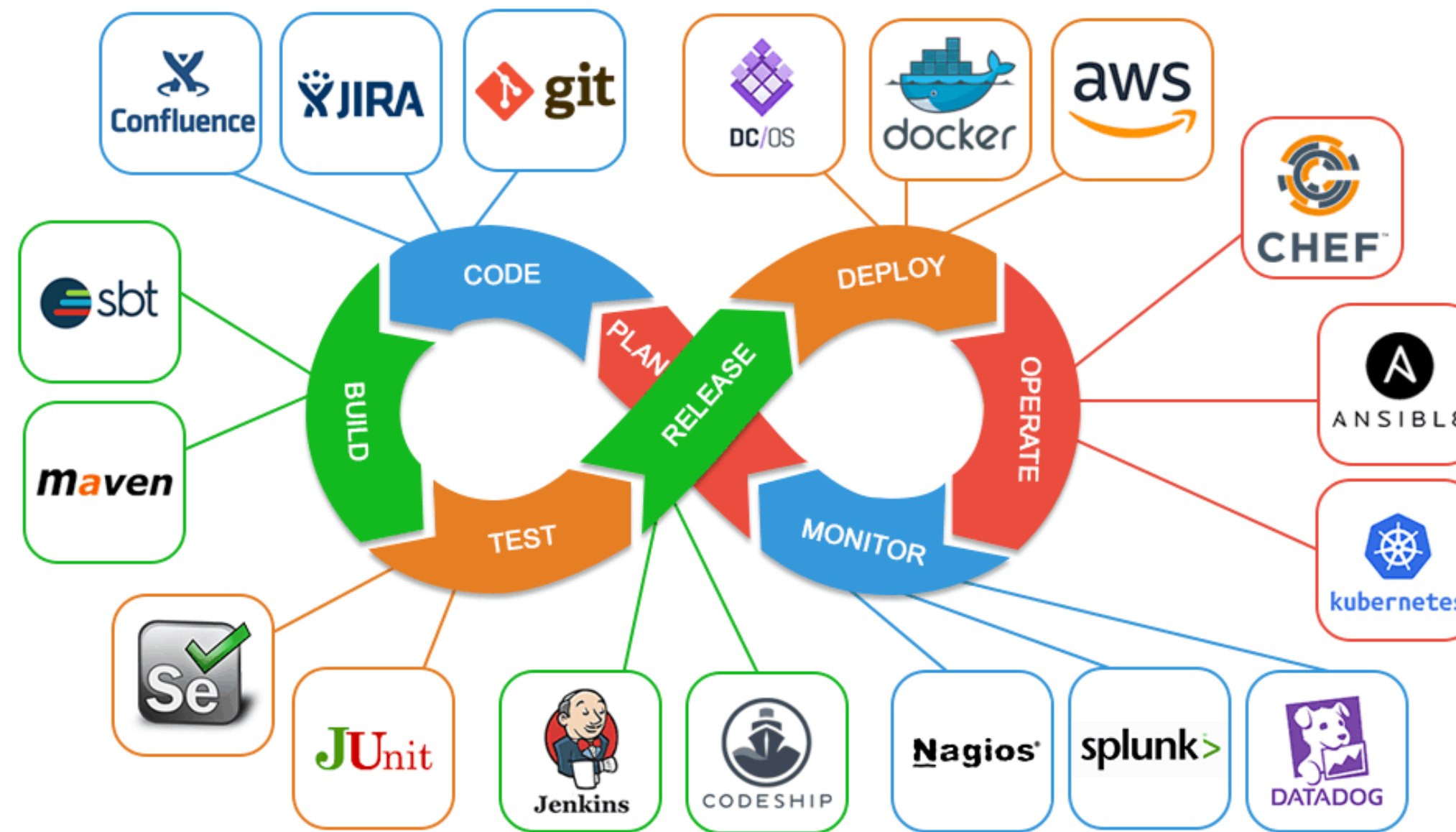
Scrum



Lean разработка



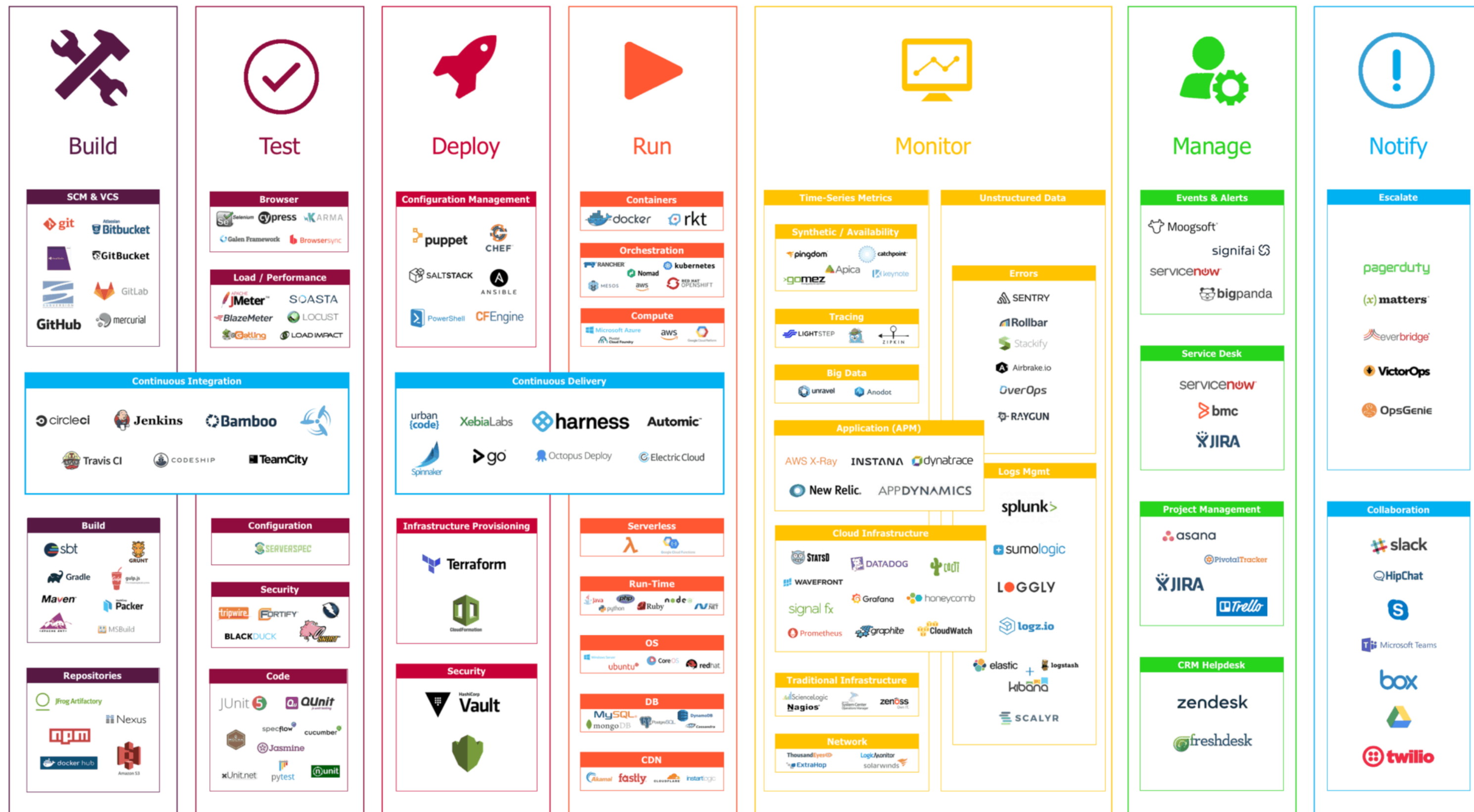
DevOps vs Agile



Разработка, тестирование и развертывание ПО:

- Agile: подразумевает, что процесс останавливается после этих трех этапов
- DevOps: включает в себя операции, которые происходят постоянно. Поэтому мониторинг и разработка ПО продолжают и после получения заказчиком готового продукта.

Инструменты жизненного цикла DevOps



Системы контроля версий



GitHub



GitLab



Perforce



Beanstalk



AWS CodeCommit



Apache Subversion



Team Foundation
Server



Mercurial

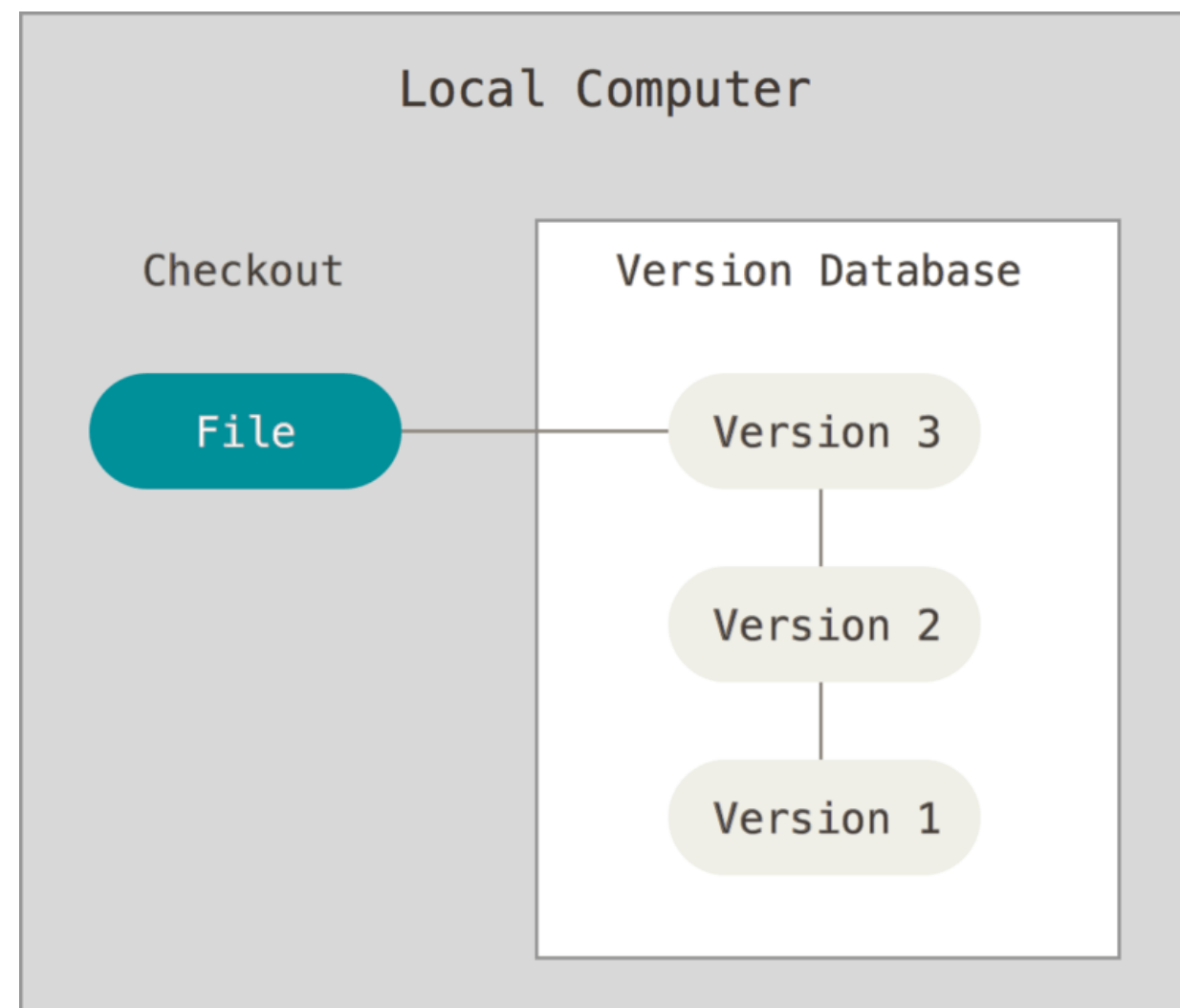


Bitbucket

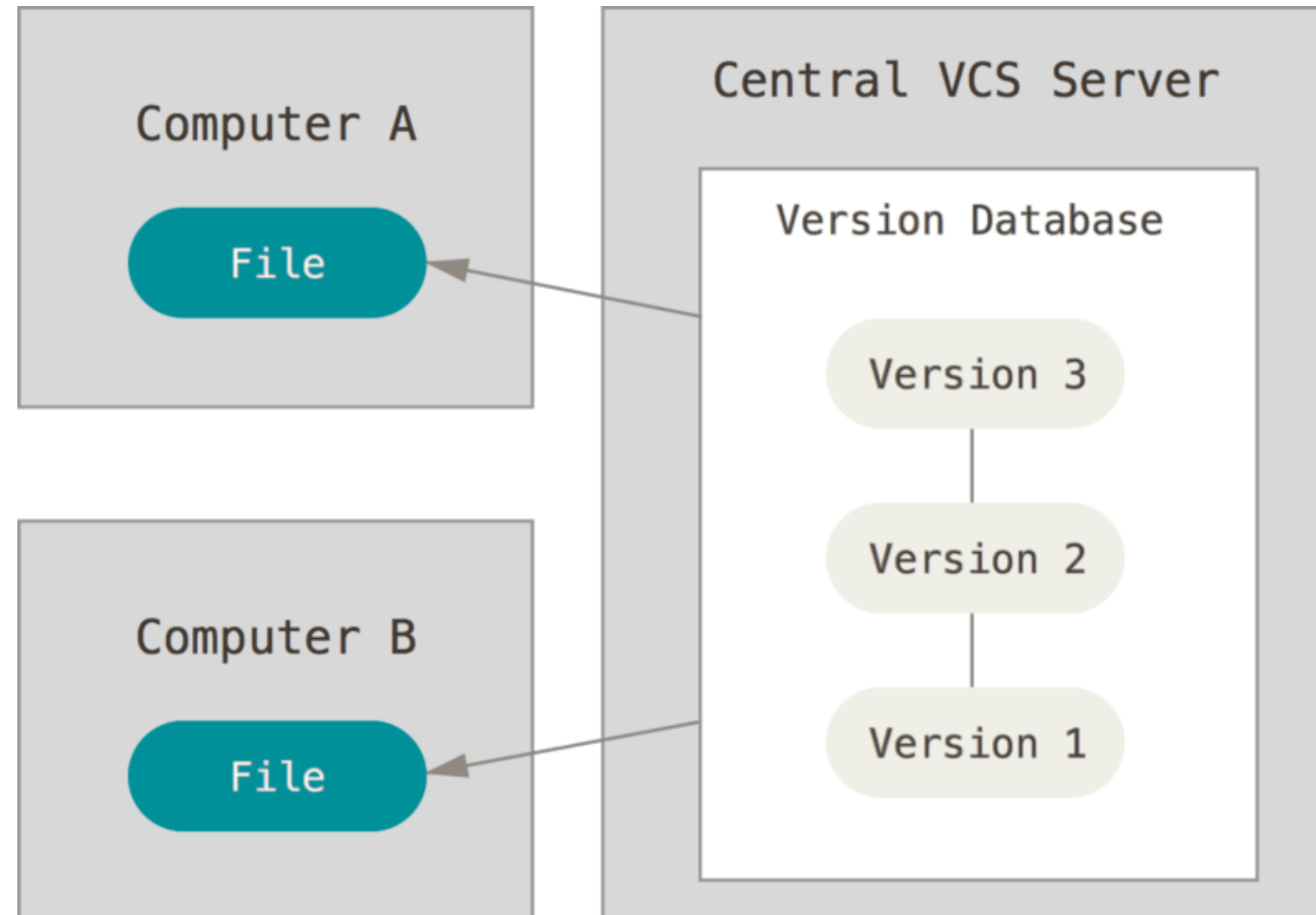


Concurrent Version Control

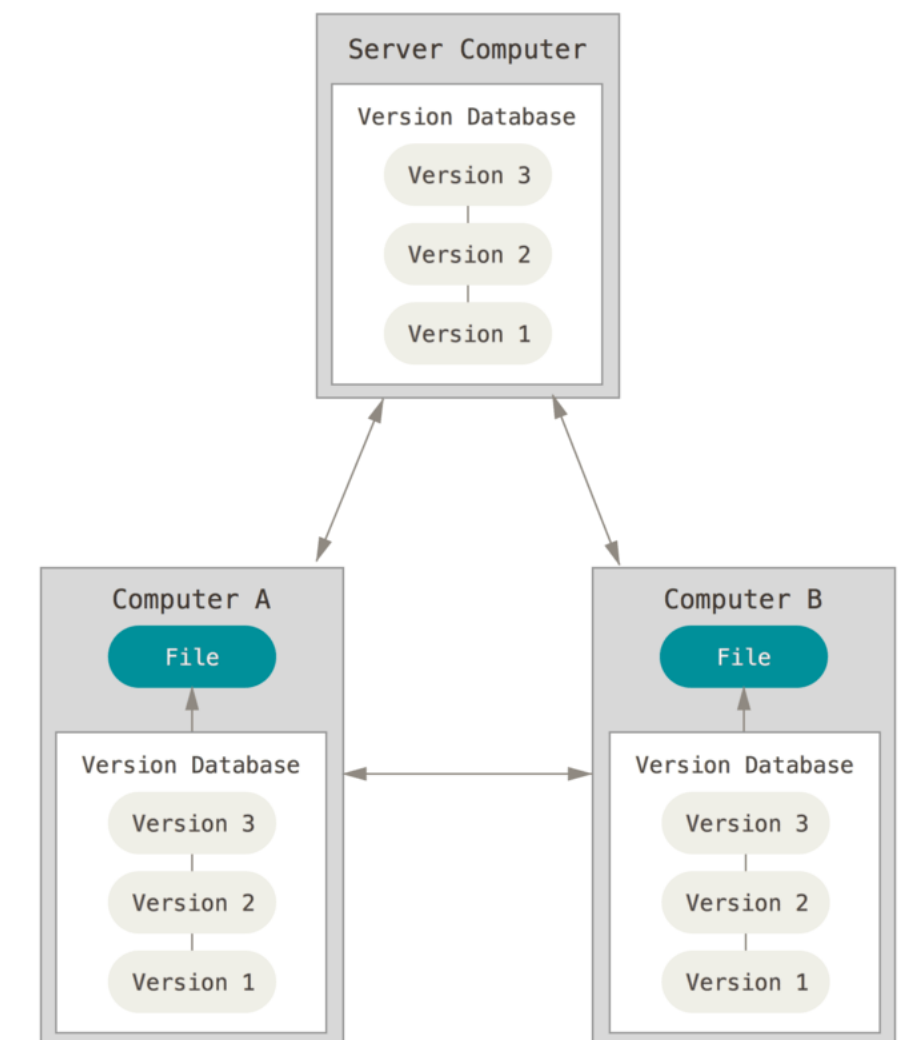
Типы систем контроля версий



Локальные
(Revision Control System)



Централизованные
(Subversion, Perforce)



Распределенные
(Git, Mercurial)

Git

- Git — распределённая система управления версиями.
- Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux.
- Первая версия выпущена 7 апреля 2005 года.

- Git is created by **Linus Torvalds**, the creator of Linux
- It was developed initially to manage the Linux development community
- Linux code has been managed,
 - 1991-2002: Using an archive of patches
 - 2002-2005: Using **BitKeeper**
 - 2005-Now: Using **Git**
- Target was,
 - Speed
 - Simple design
 - Fully distributed
 - Strong support for non-linear development (thousands of parallel branches)
 - Able to handle large projects like the Linux kernel efficiently

Как работать с Git

- консольный клиент
- графические клиенты
- встроенные в IDE
- Web IDE



GIT – конфигурация

Файл **/etc/gitconfig** содержит значения, общие для всех пользователей системы и для всех их репозиторий. Если при запуске `git config` указать параметр `--system`, то параметры будут читаться и сохраняться именно в этот файл.

Файл **~/.gitconfig** или **~/.config/git/config** хранит настройки конкретного пользователя. Этот файл используется при указании параметра `--global`

Файл **config** в каталоге Git (т.е. **.git/config**) репозитория, который вы используете в данный момент, хранит настройки конкретного репозитория. Этот файл используется при указании параметра `--local`.

Конфигурация:

```
$ git config --global user.name <name>
$ git config --global user.email <e-mail>
$ git config --list
```

Как работает Git

Git — простое хранилище типа ключ-значение:

- В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение
- Обращение к сохранённым объектам происходит по хеш-сумме
- Невозможно изменить содержимое файла или директории так, чтобы Git не узнал об этом



Какие проблемы решает Git?

- Откат к ранним версиям проекта
- Хранение истории и отслеживание изменений
- Организация совместной работы над проектом

Хранение изменений в Git

Слепки вместо патчей

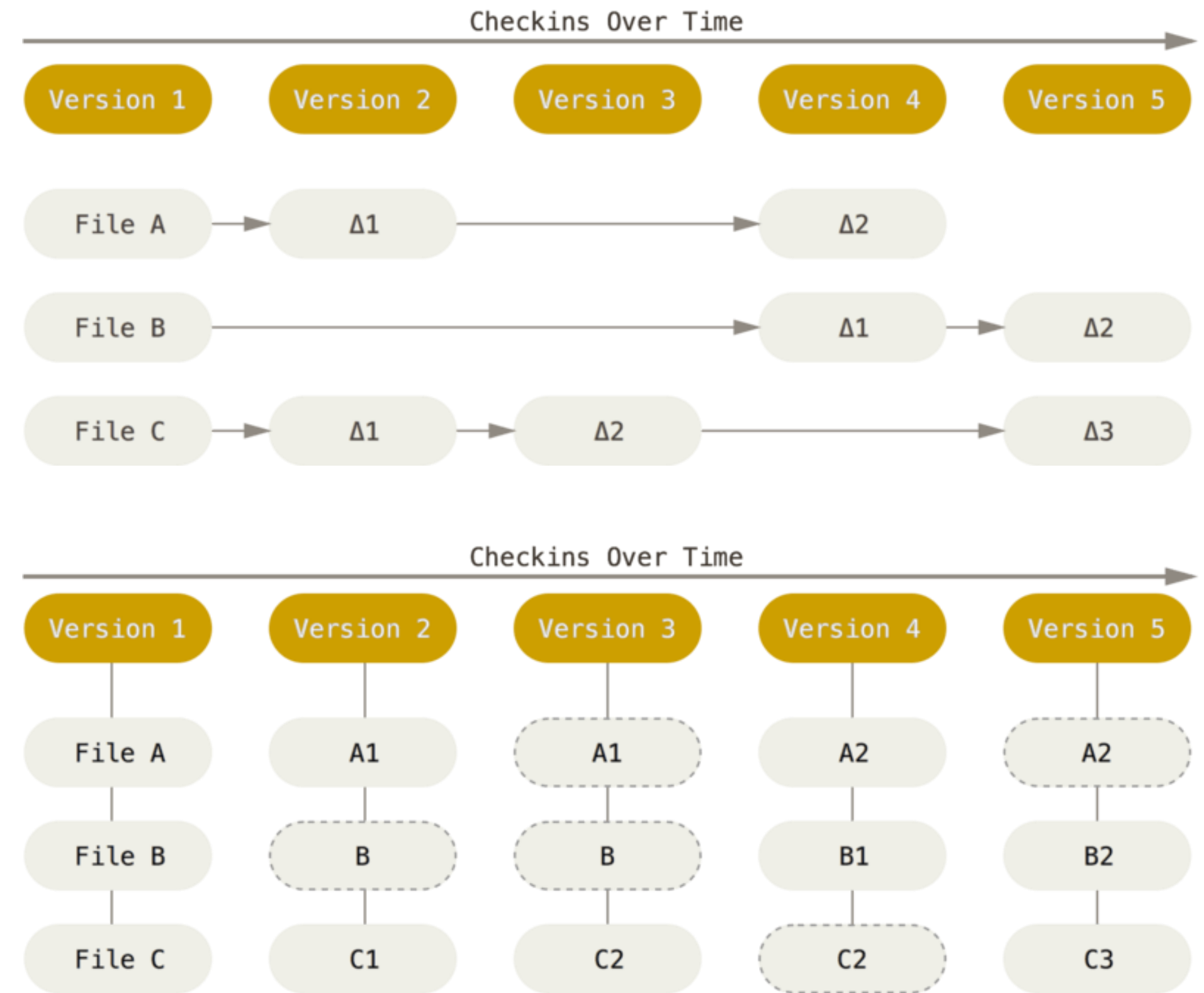
Большинство систем контроля версий хранит информацию как список изменений для файлов.

Вместо этого Git считает хранимые данные набором слепков небольшой файловой системы.

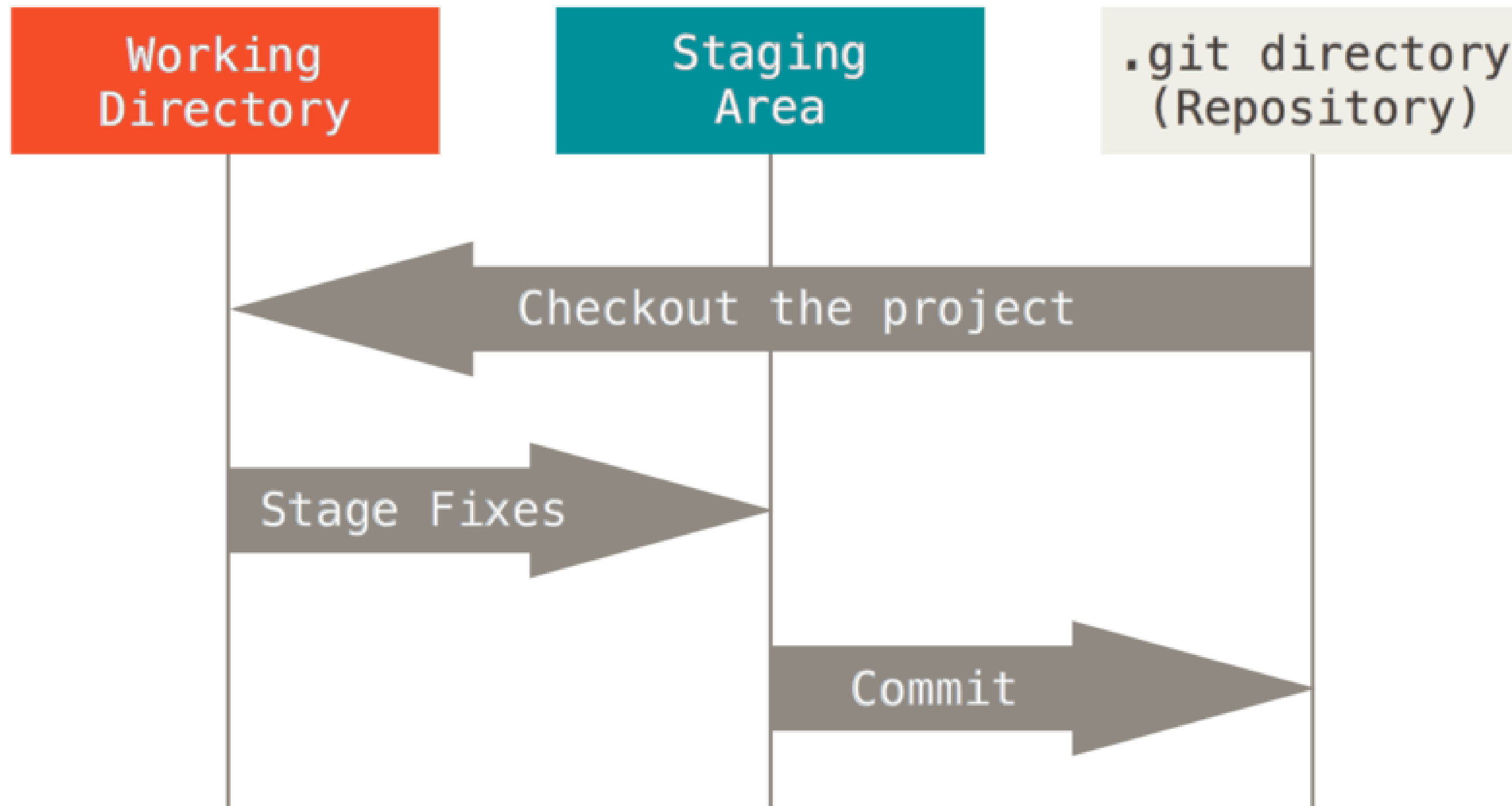
Когда вы фиксируете текущую версию проекта, Git сохраняет слепок того, как выглядят все файлы проекта на текущий момент.

Если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохраненный файл.

На самом деле всё сложнее и есть так называемые Раск-файлы



GIT – три рабочие области



GIT – структура хранения

```
.git
├── COMMIT_EDITMSG
├── FETCH_HEAD
├── HEAD
├── config
├── description
├── hooks
├── index
├── info
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
├── objects
│   ├── 01
│   │   └── 2dc51bccc1
│   ├── 5a
│   │   └── 2669f47b80
│   ├── 7b
│   │   ├── bob3154035
│   │   └── cf9b7d210c
│   ├── 9d
│   │   └── 4fc2f6782e
│   ├── f7
│   │   └── a9d2e60188
│   ├── fd
│   │   └── 96eb24c77b
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
```

Все данные расположены в директории `.git` в корне проекта
Создать пустой репозиторий можно командой: `git init`.

В репозитории содержатся:

- объекты коммитов
- ссылки на коммиты и ветки
- конфигурация
- скрипты-хуки

Ключевые элементы Git:

- каталог `objects` - база данных объектов Git
- каталог `refs` - ссылки на объекты коммитов в базе
- файл `HEAD` - указатель на текущий коммит
- файл `index` - хранит содержимое индекса
- файл `config` – файл конфигурации репозитория

GIT – структура хранения

Концептуально, данные хранятся в Git примерно в виде дерева:

BLOB – содержит длину содержимого файла и само содержимое

TREE – хранит список записей, который соответствуют иерархии файловой системы

COMMIT – хранит ссылку на объект TREE и ссылку на родительский коммит

TAGS - различают два типа:

- Легковесные теги - указатели на определенный коммит
- Аннотированные - хранятся как полноценные объекты

Посмотреть тип объекта:

```
$ git cat-file -t <ХЭШ_ОБЪЕКТА>
```

Посмотреть содержимое объекта:

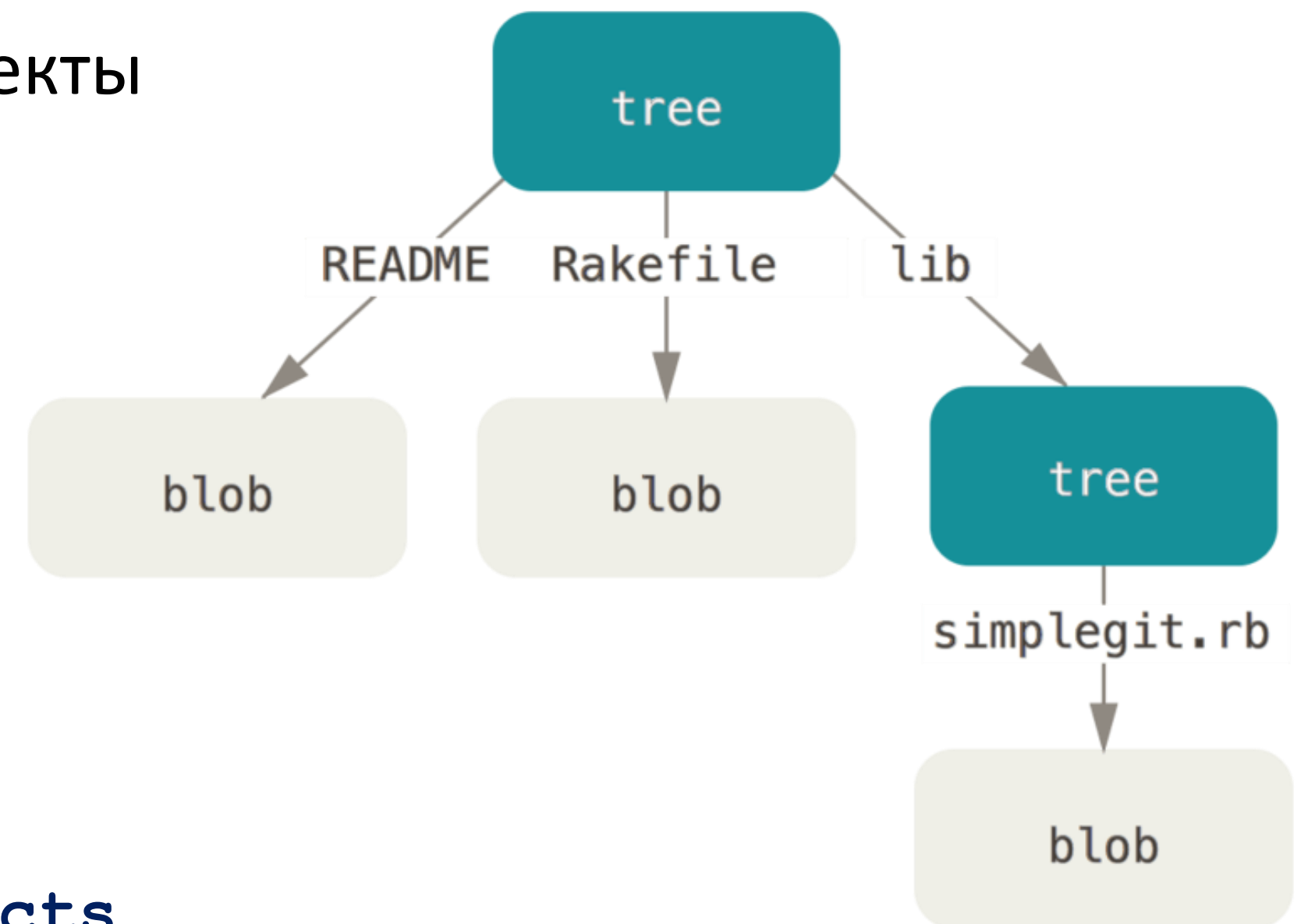
```
$ git cat-file -p <ХЭШ_ОБЪЕКТА>
```

Посмотреть куда ссылается tree:

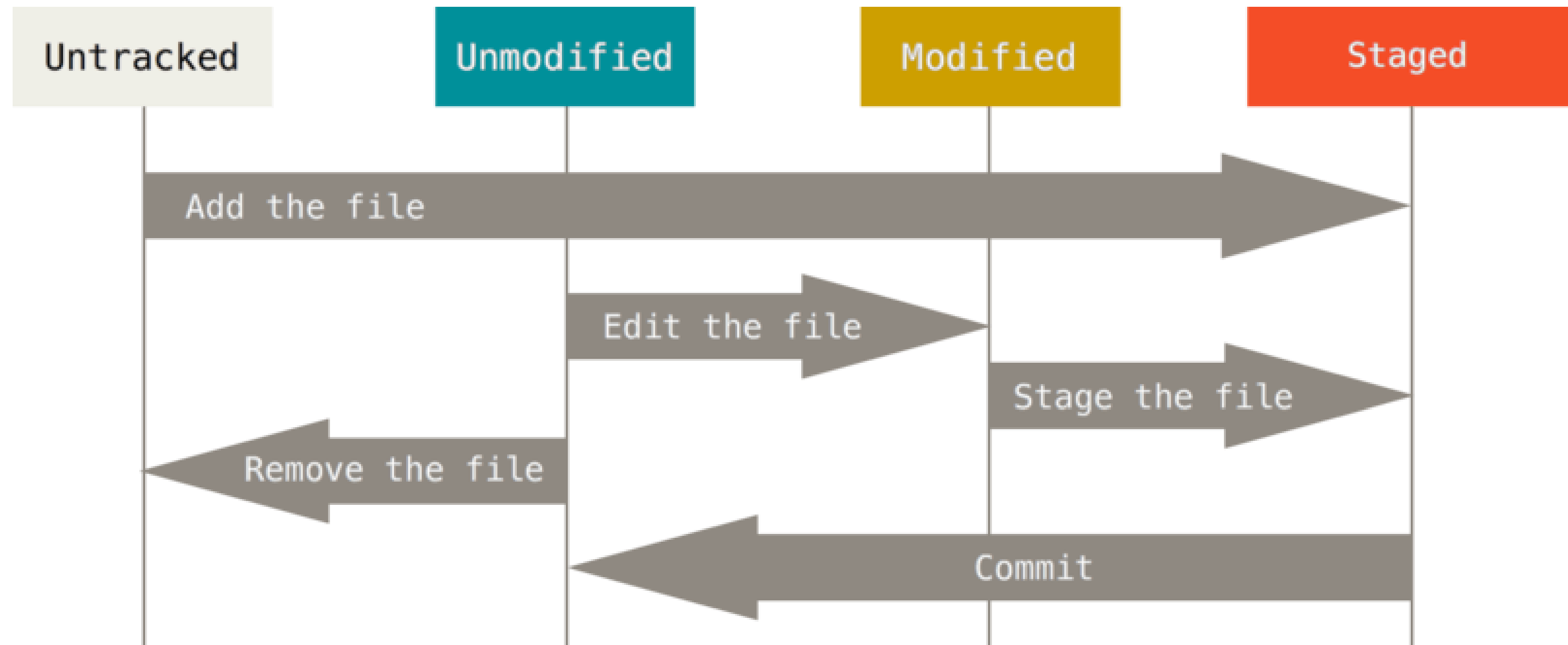
```
$ git ls-tree <ХЭШ_ОБЪЕКТА>
```

Посмотреть все сущности:

```
$ git cat-file --batch-check --batch-all-objects
```



GIT – состояния файлов



Основной инструмент определения состояний — это команда **git status**

GIT – работа с ветками

ВЕТКА — это простой файл, содержащий 40 символов контрольной суммы SHA-1 коммита, на который она указывает

`$ git branch` ⇒ покажет все ветки

`$ git branch <branch name>` ⇒ создаст новую ветку

`$ git checkout <branch name>` ⇒ переключить указатель на новую ветку

`$ git checkout -b <branch name>` ⇒ создать новую ветку и переключиться на нее

HEAD — это указатель на текущую локальную ветку

HEAD перемещается командой:

`$ git checkout <ХЭШ>`

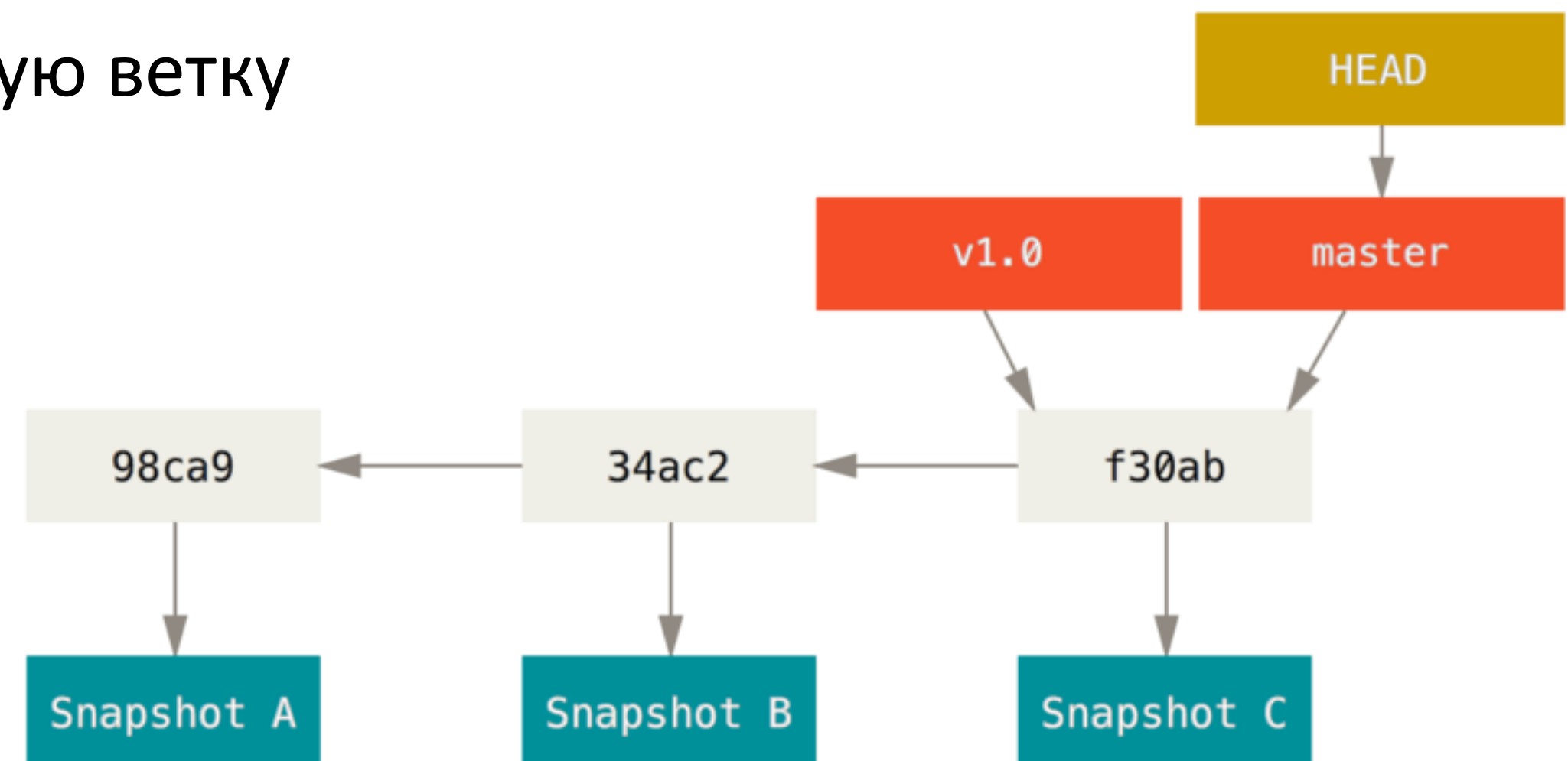
или через относительный указатель

`$ git checkout HEAD~<число прыжков>`

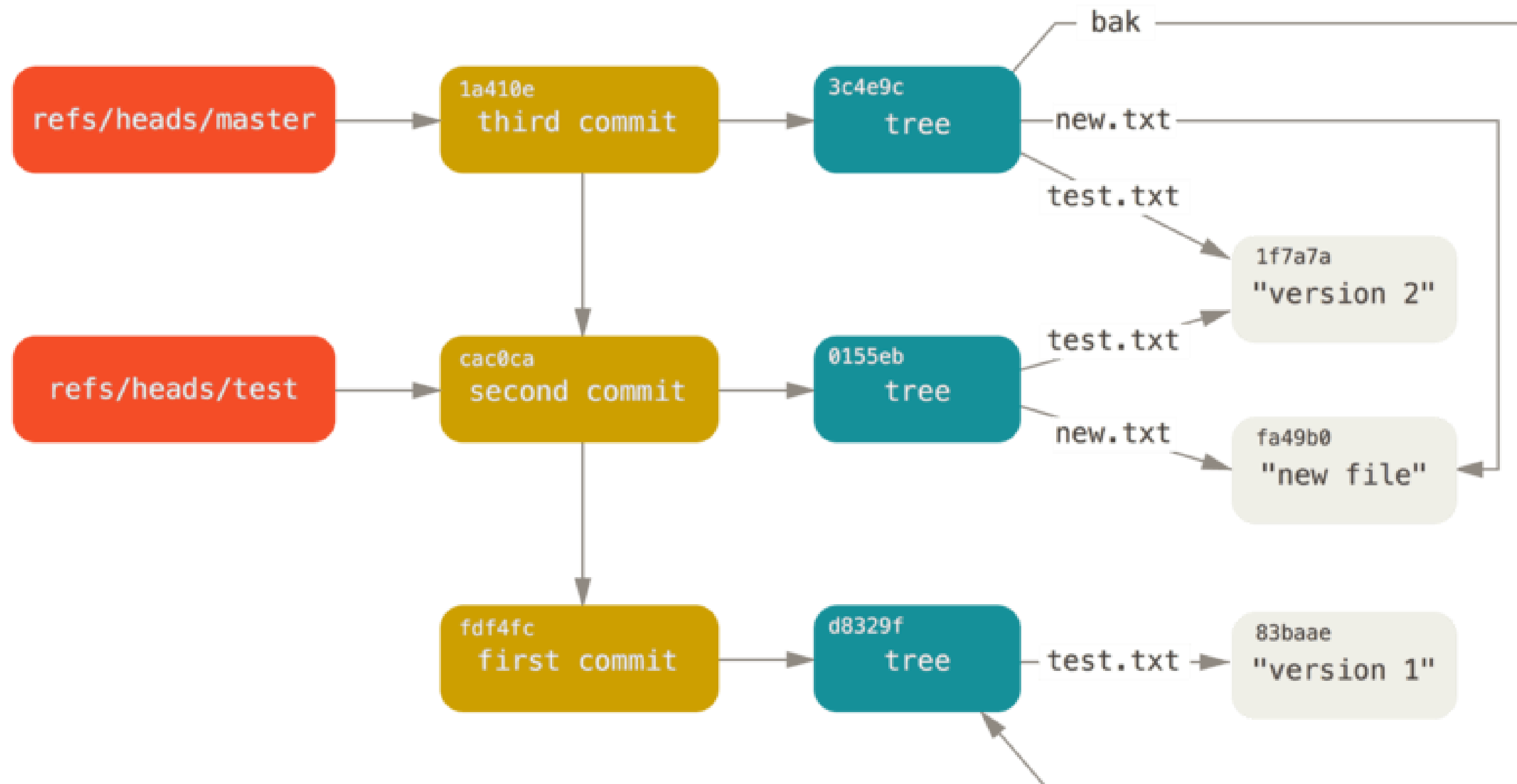
Например:

`$ git checkout d251a9d`

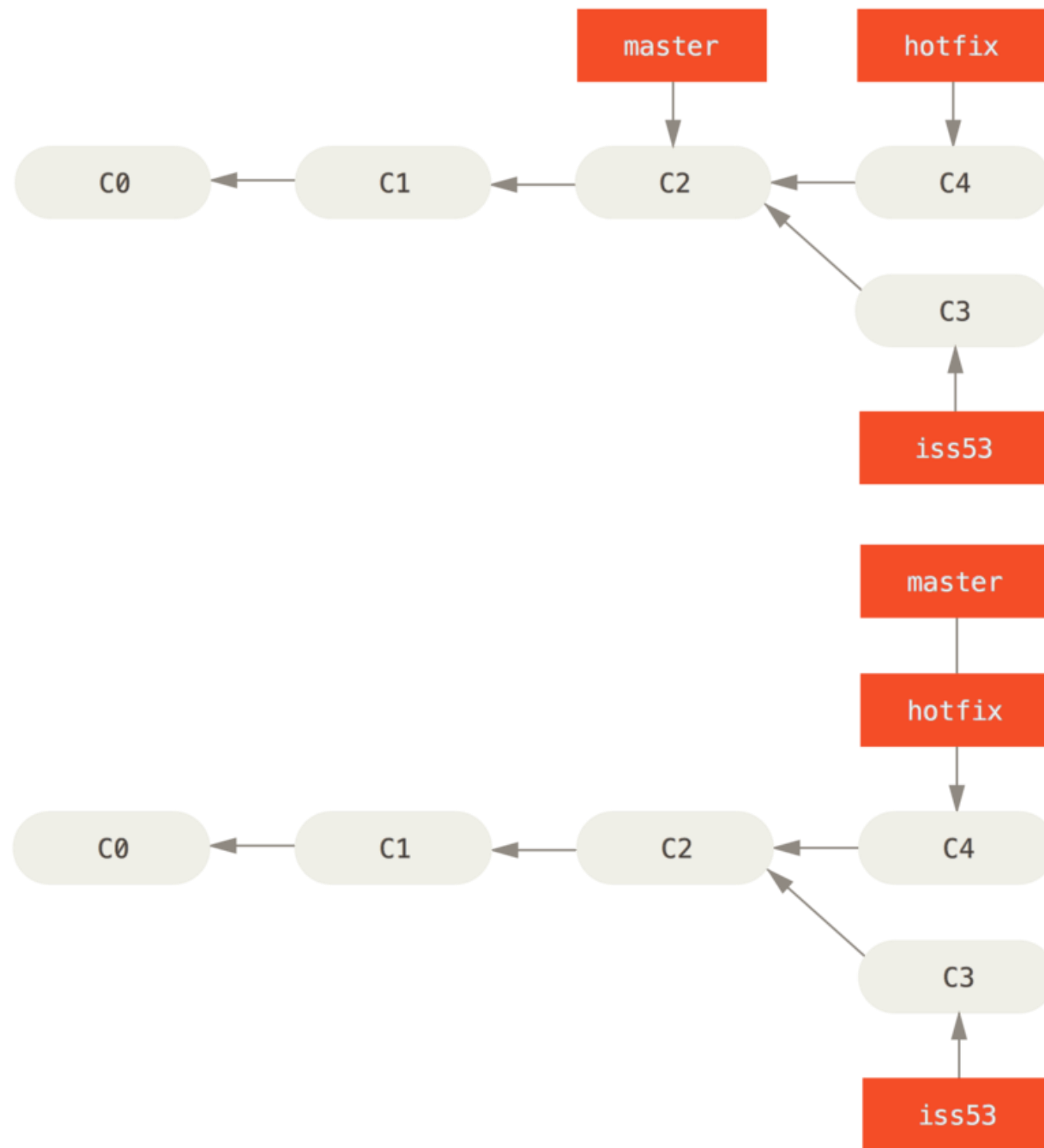
`$ git checkout HEAD~2`



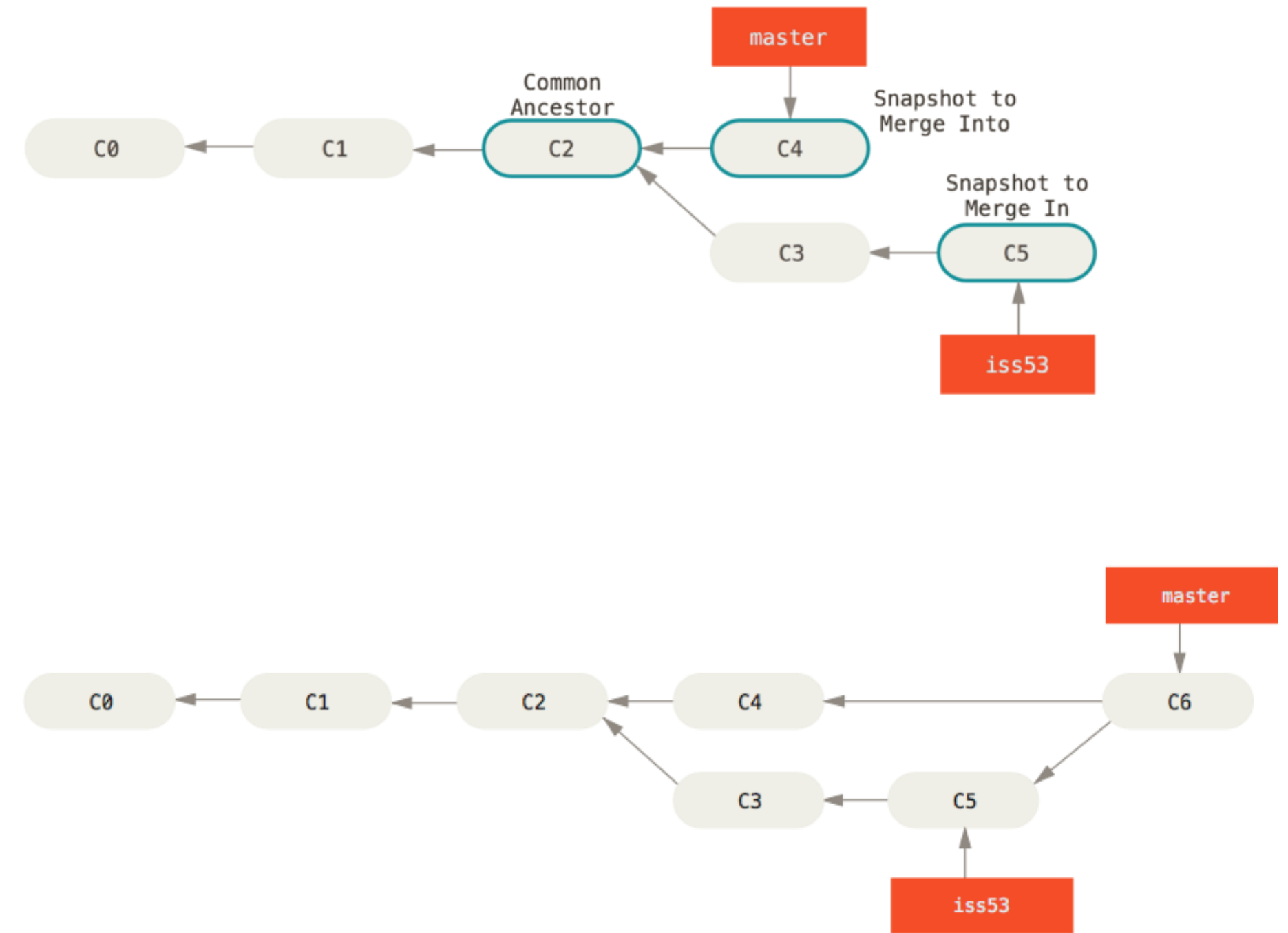
GIT – работа с ветками



GIT – слияние веток. Merge



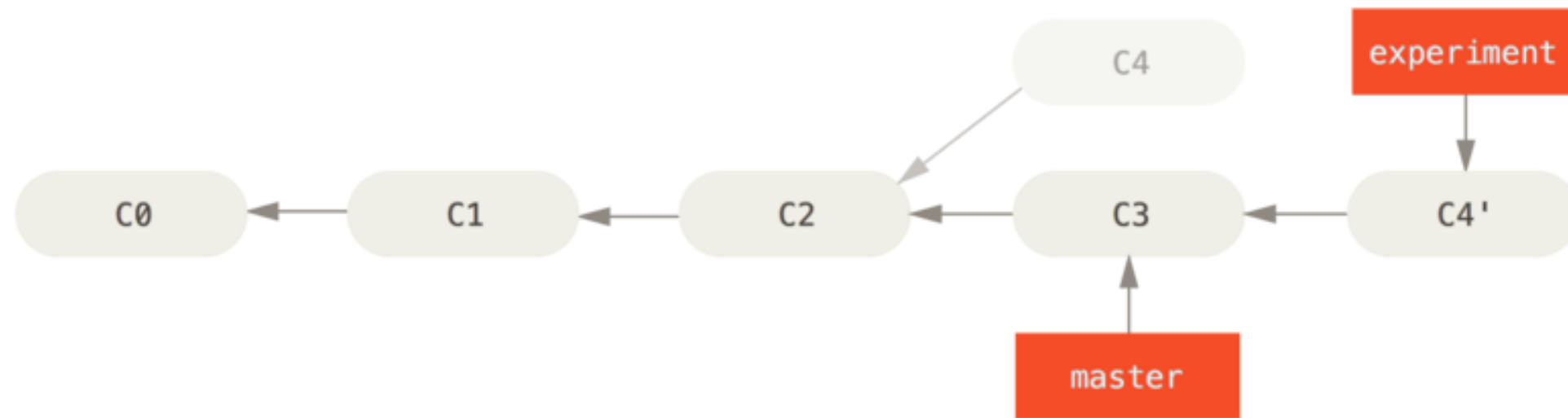
Fast-forward



Коммит слияния

`$ git merge <ветка>` ⇒ сливает указанную ветку в текущую

GIT – слияние веток. Rebase

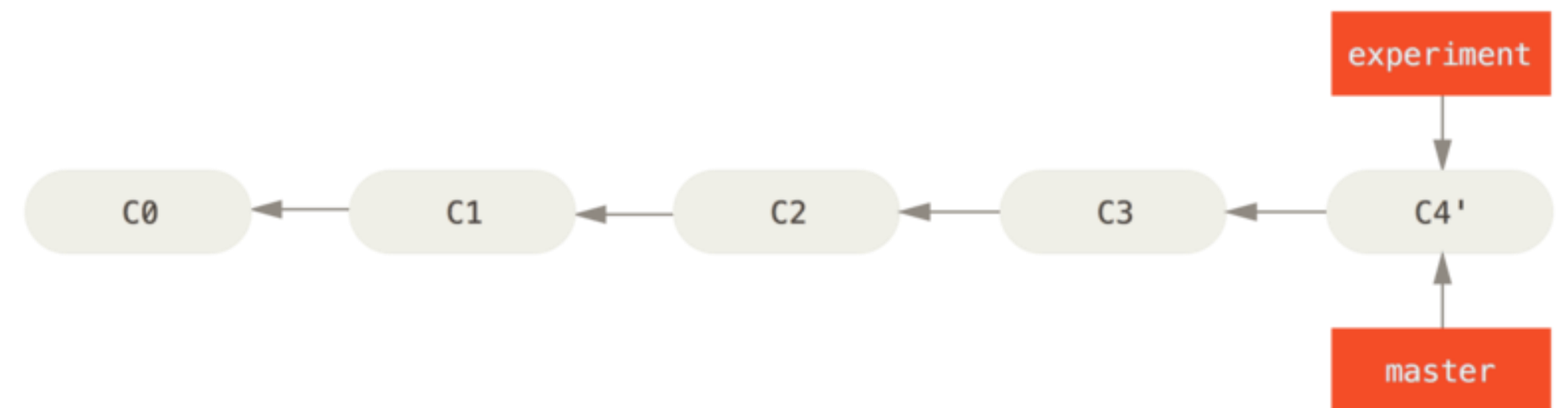


```
$ git checkout experiment
```

```
$ git rebase master
```

First, rewinding head to replay your work on top of it...

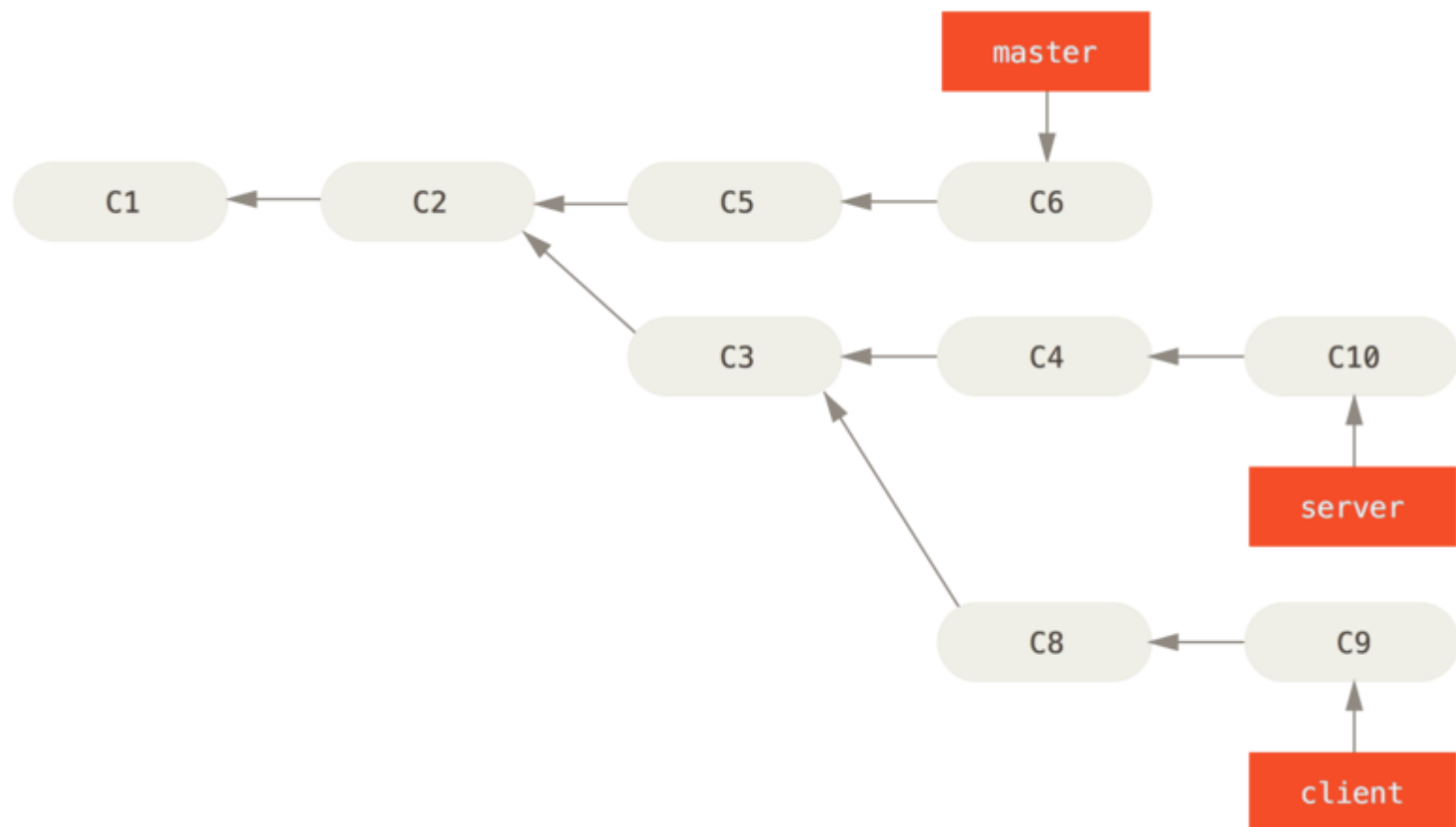
Applying: added staged command



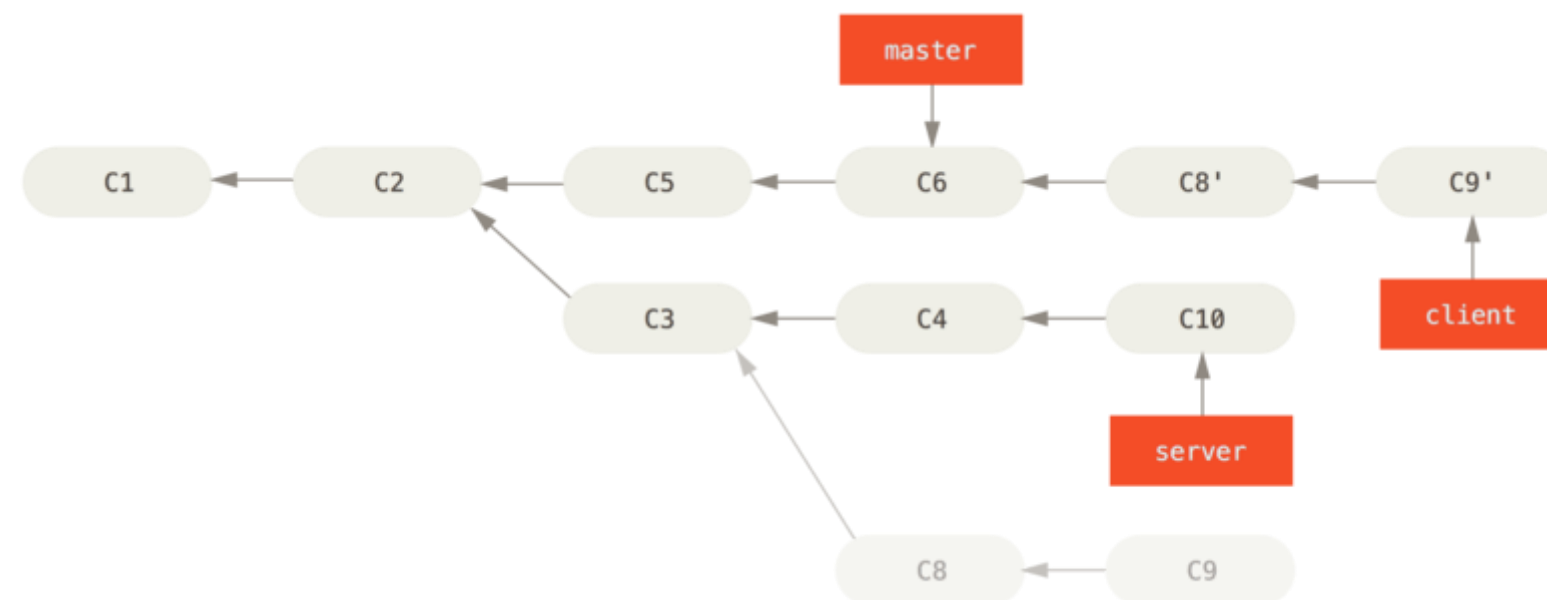
```
$ git checkout master
```

```
$ git merge experiment
```

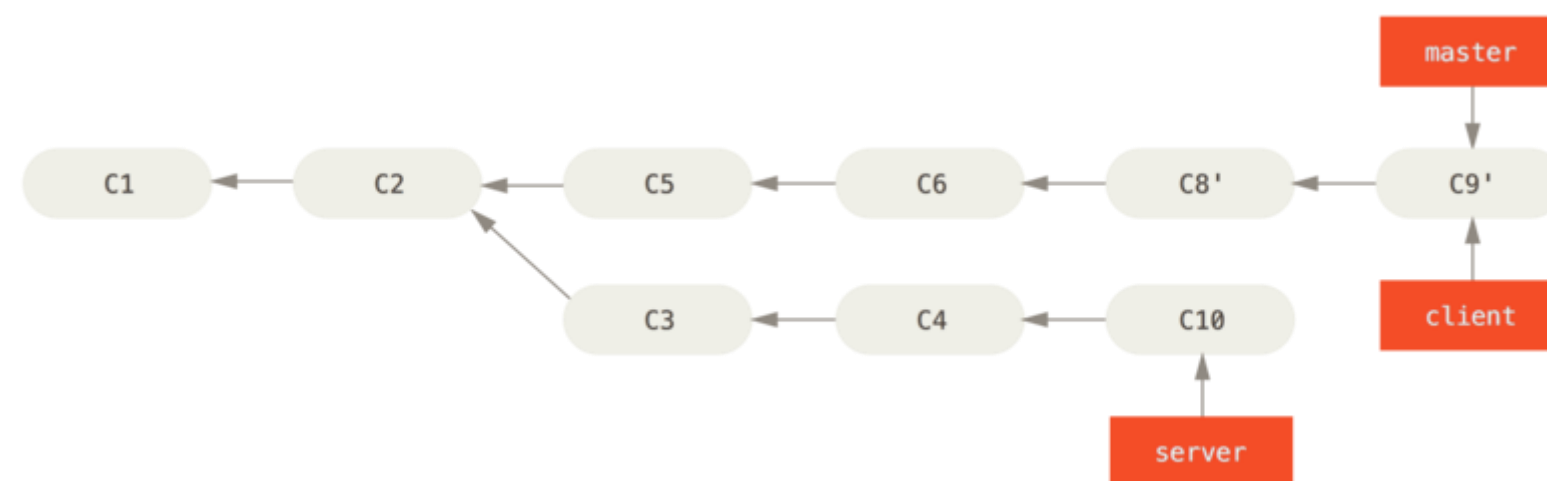

GIT – слияние веток. Rebase



- Создали тематическую ветку **server**, чтобы добавить в проект некоторую функциональность для серверной части, и делаете коммит.
- Выполнили ответвление в **client**, чтобы сделать изменения для клиентской части, и создали несколько коммитов. Вернулись на ветку **server** и сделали ещё несколько коммитов.
- Решили внести изменения клиентской части в основную ветку **master** для релиза, но при этом не хотите добавлять изменения серверной части до полного тестирования.



\$ git rebase --onto master server client
«Переключись на ветку **client**, найди изменения относительно ветки **server** и примени их для ветки **master**».



\$ git checkout master
\$ git merge client

Merge vs Rebase

Потенциальные недостатки Rebase:

- Разрешение конфликтов в случае rebase сложнее
- Перебазирование публичных веток может быть опасным при работе в команде

Когда использовать перебазирование, а когда слияние?

Не используйте перебазирование:

- Если ветка является публичной. Переписывание общих веток будет мешать работе других членов команды.
- Когда важна точная история коммитов ветки (так как rebase переписывает историю).

Предпочтительно использовать:

- **rebase** для кратковременных, локальных веток,
- **merge** для веток в публичном репозитории.

GIT – отмена изменений

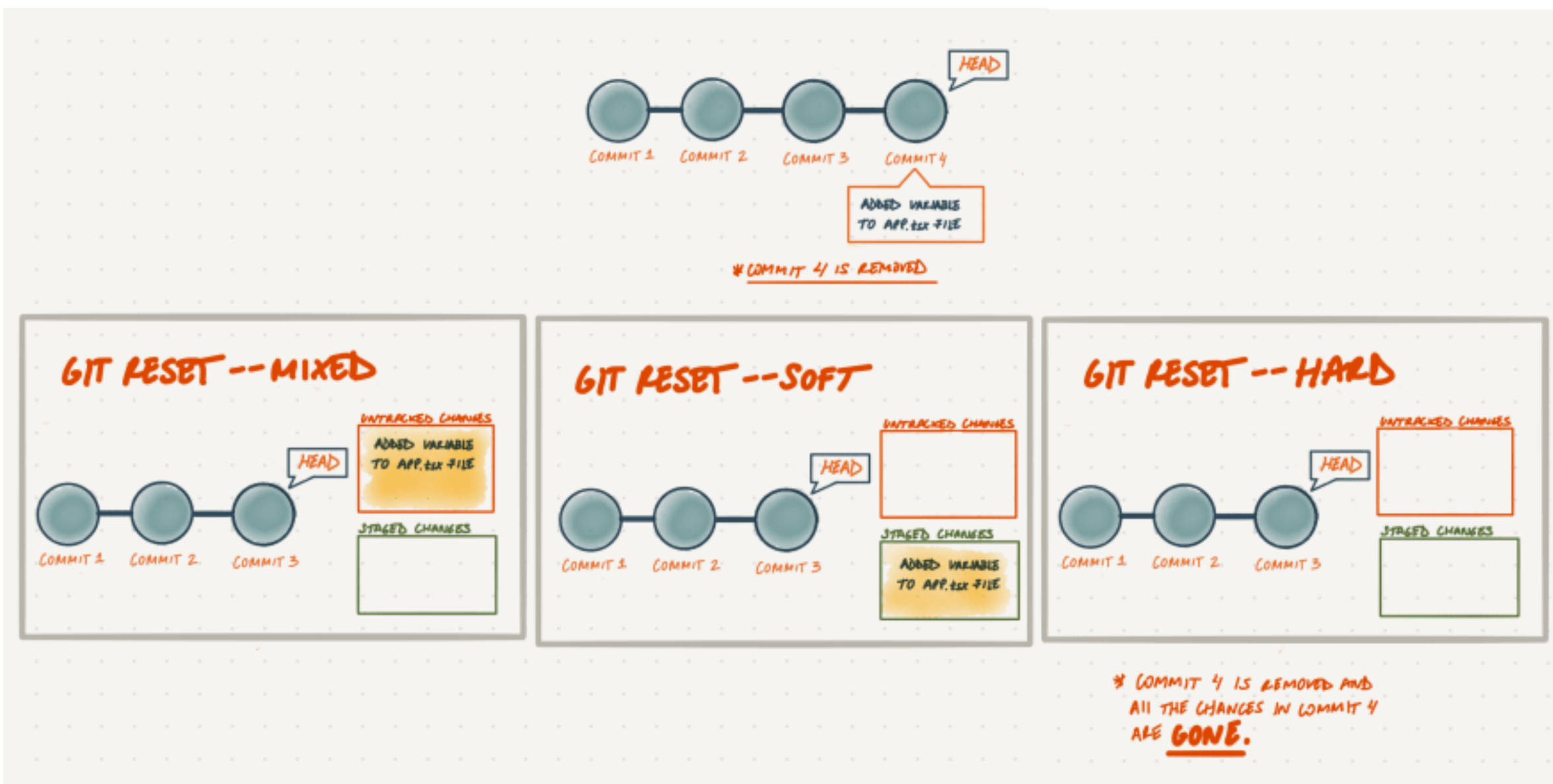
git reset —инструмент для отмены изменений.

Имеет три основные режима:

\$ **git reset --soft** ⇒ Обновление HEAD

\$ **git reset --mixed** ⇒ Обновление индекса

\$ **git reset -hard** ⇒ Обновление рабочей директории



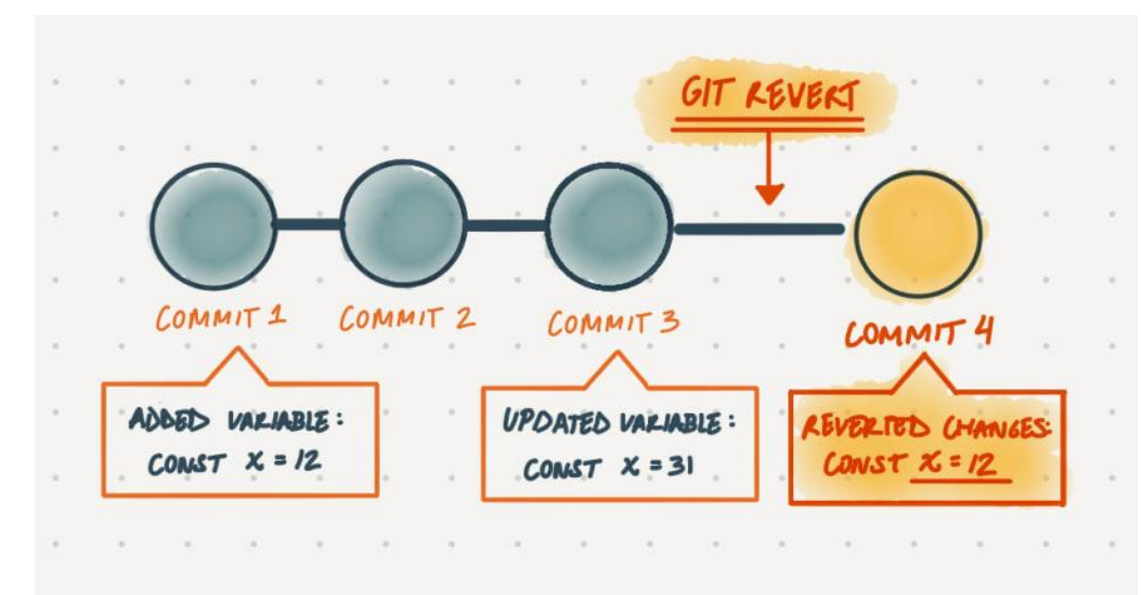
git revert — безопасный способ отменить изменения. Отменяет действия прошлых коммитов, создавая новый, содержащий все отменённые изменения.

С помощью commit-хэшей:

\$ **git revert [SHA]**

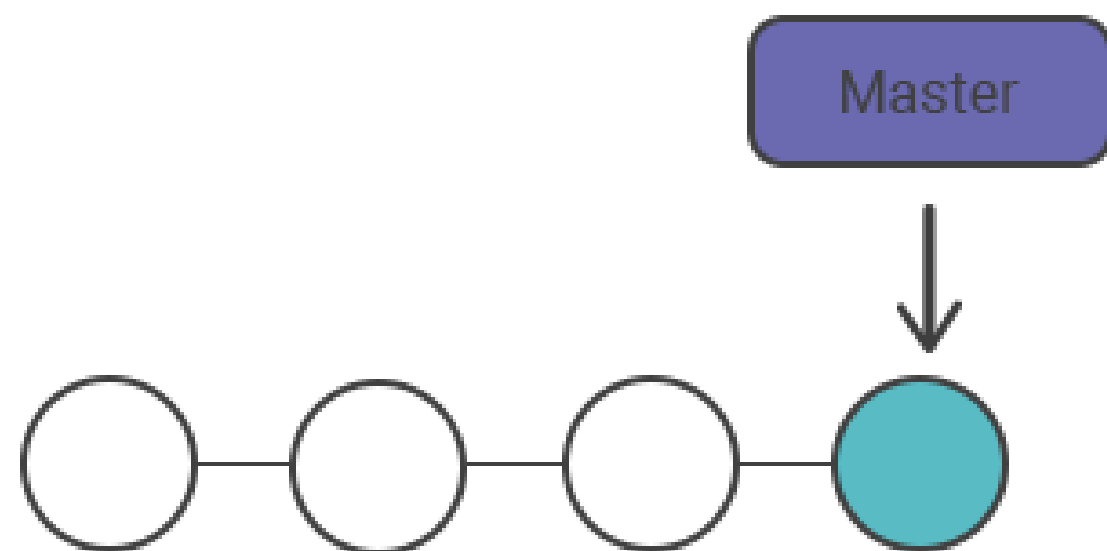
С помощью диапазонов:

\$ **git revert HEAD~[num-of-commits-back]**



GIT – изменение последнего коммита

git commit --amend — это удобный способ изменить последний коммит. Она позволяет объединить проиндексированные изменения с предыдущим коммитом без создания НОВОГО КОММИТА.

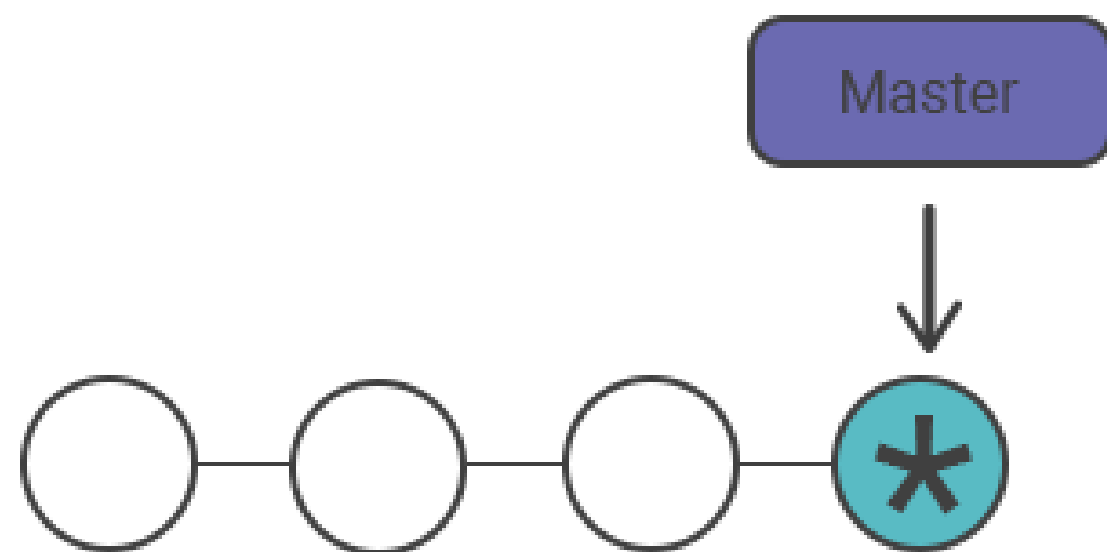


Изменение сообщения коммита:

```
$ git commit --amend -m "an updated commit message"
```

Изменение файлов в коммите без изменения сообщения:

```
$ git commit --amend --no-edit
```



Не используйте amend для публичных коммитов

Измененные коммиты по сути являются новыми коммитами. При этом предыдущий коммит не останется в текущей ветке. Не изменяйте коммит, после которого уже начали работу другие разработчики.

GIT – изменение нескольких коммитов

В случаях, когда важно сохранить чистую историю проекта, добавление флага **-i** к команде **git rebase** позволяет выполнять интерактивную операцию rebase. Это дает возможность изменять отдельные коммиты в процессе, а не перемещать все коммиты.

Изменить последние три коммита:

```
$ git rebase -i HEAD~3
```

Каждый коммит, входящий в диапазон HEAD~3..HEAD, будет изменён.

Не включайте в такой диапазон коммит, который уже был отправлен в публичный репозиторий.

```
pick f7f3f6d Change my name a bit
pick 310154e Update README formatting and add blame
pick a5f4a0d Add cat-file

# Rebase 710f0f8..a5f4a0d onto 710f0f8
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

GIT – объединение коммитов

С помощью интерактивного режима команды **git rebase** также можно объединить несколько коммитов в один.

```
$ git rebase -i HEAD~3
```

Если вместо «pick» или «edit» указать «squash», Git применит изменения из текущего и предыдущего коммитов и предложит вам объединить их сообщения.

Таким образом, если вы хотите из этих трёх коммитов сделать один, вы должны изменить скрипт следующим образом:

```
pick f7f3f6d Change my name a bit
squash 310154e Update README formatting and add blame
squash a5f4a0d Add cat-file
...
```


GIT – удаление файла из каждого коммита

Если требуется удалить какой-то файл из всех коммитов можно воспользоваться командой **git filter-branch**, чтобы привести к нужному виду всю историю.

Для удаления файла **passwords.txt** из всей истории можете использовать опцию **--tree-filter** команды **filter-branch**:

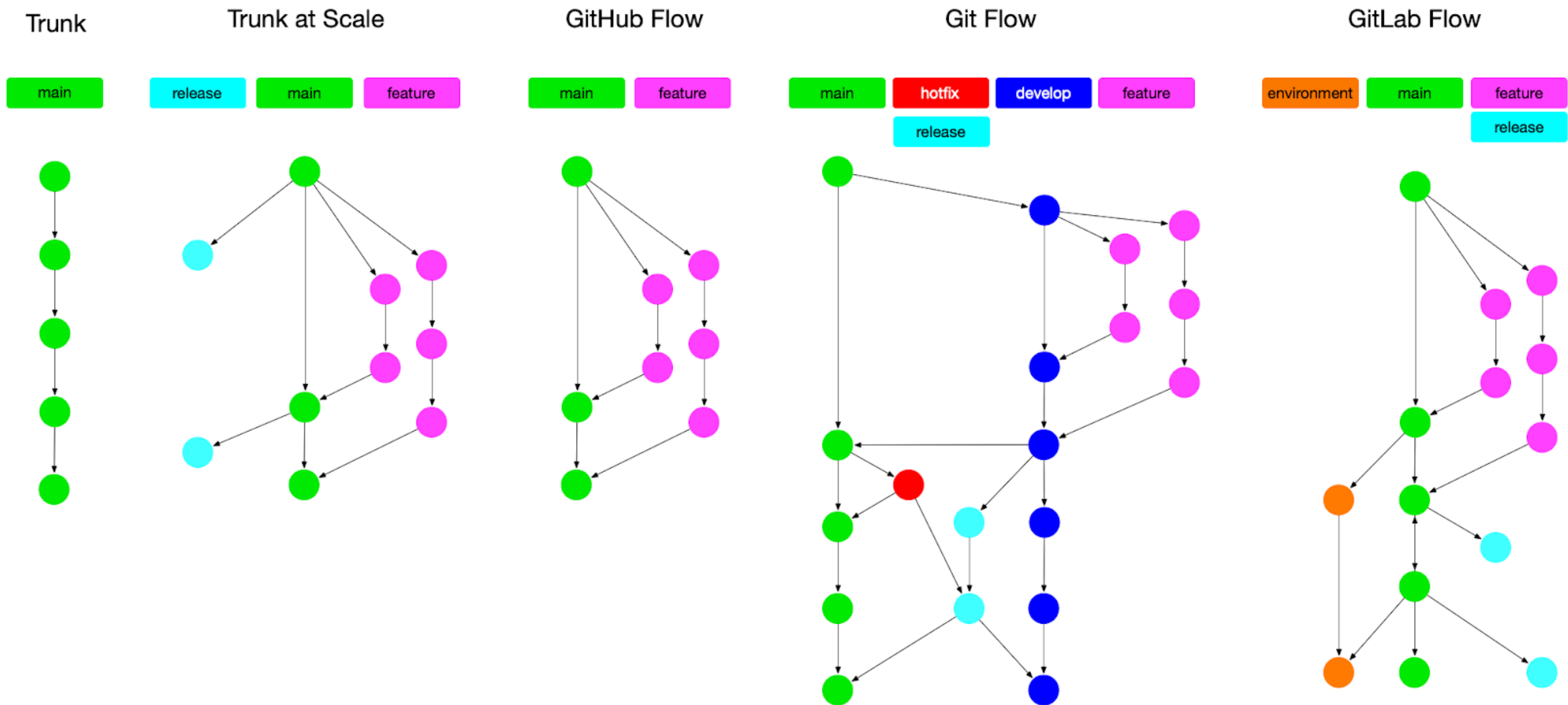
```
$ git filter-branch --tree-filter 'rm -f passwords.txt' HEAD
Rewrite 6b9b3cf04e7c5686a9cb838c3f36a8cb6a0fc2bd (21/21)
Ref 'refs/heads/master' was rewritten
```

Опция **--tree-filter** выполняет указанную команду после переключения на каждый коммит и затем повторно фиксирует результаты.

Для выполнения **filter-branch** на всех ваших ветках, вы можете передать команде опцию **--all**.

Хорошим подходом будет выполнение всех действий в тестовой ветке и, после проверки полученных результатов, установка на неё указателя основной ветки.

GIT – модели разработки



For a Smile ;)

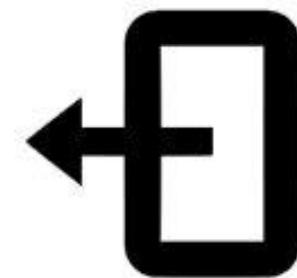
In case of fire



1. `git commit`



2. `git push`



3. `leave building`

Continuous Integration / Continuous Delivery

CI/CD PIPELINE



CI/CD

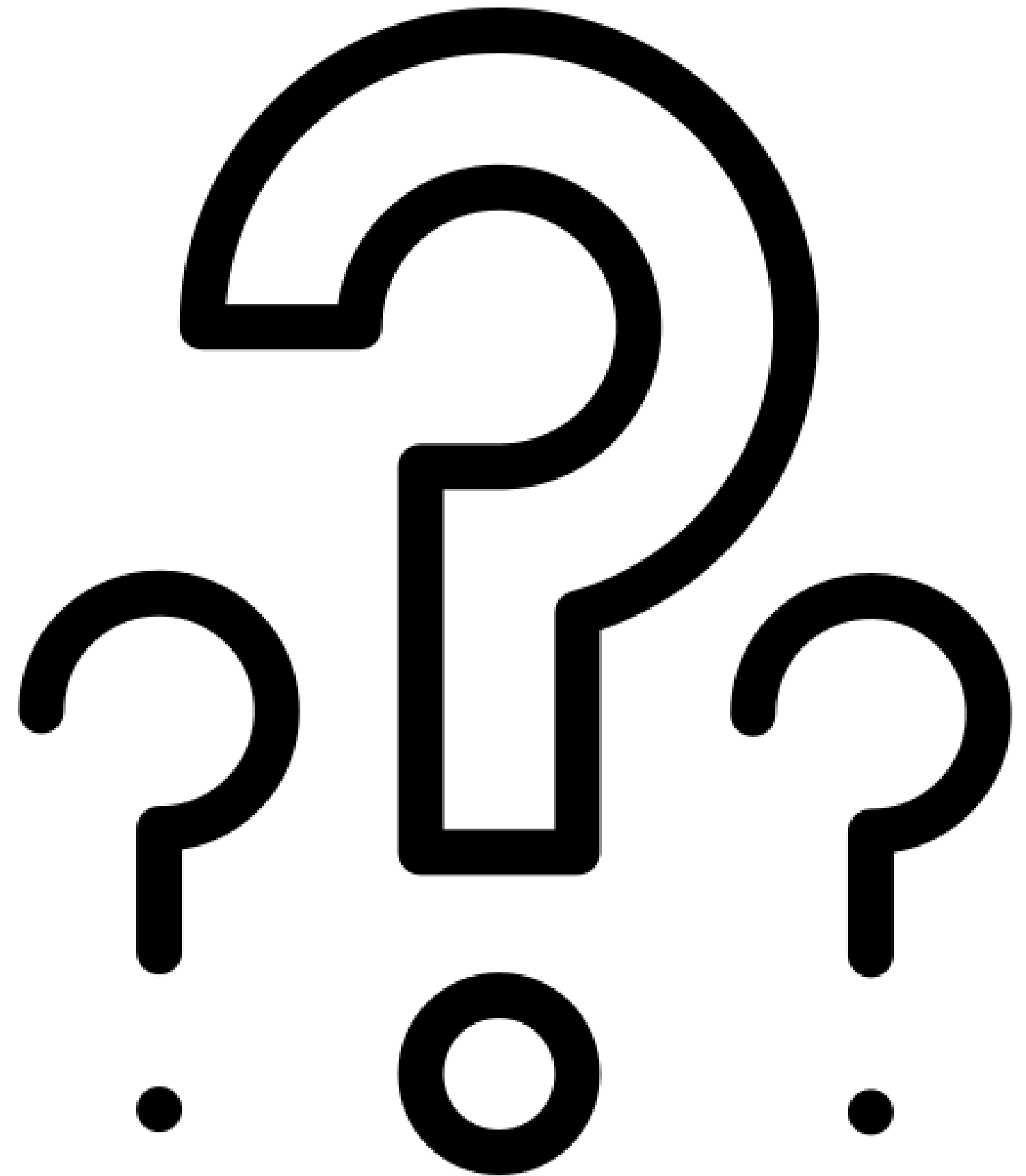


CI системы

- 2005, Hudson CI
- 2006, JetBrains TeamCity
- 2007, Atlassian Bamboo
- 2010, Hudson/Jenkins split
- 2011, Travis CI
- 2012, Gitlab
- 2019, GitHub Actions



Вопросы



**Спасибо
за внимание!**

Алексей Рагузин,
наставник,
Яндекс.Практикум

a.raguzin83@yandex.ru