Разработка научных приложений

Немного теории о Git. Удаленная работа с Git. Сервис Github.

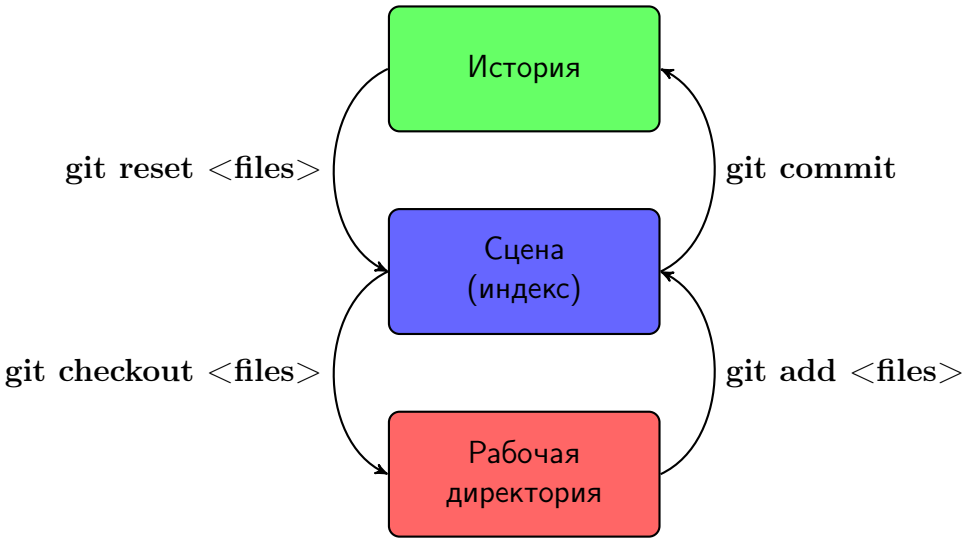
Выполнение заданий, предлагаемых ниже, поможет лучше понять, как работает система контроля версий Git. Крайне рекомендуется при работе над заданиями использовать консольный файловый менеджер Far или подобный ему.

1 Как работают основные команды Git

Следующие четыре команды предназначены для копирования файлов между рабочей директорией, сценой (так же известной как «индекс») и историей (представленной в форме коммитов).

- **git add <файлы>** — копирует файлы (в их текущем состоянии) на сцену.
- **git commit** — сохраняет снимок сцены в виде коммита.
- **git reset <файлы>** — восстанавливает файлы на сцене, а именно копирует файлы из последнего коммита на сцену. Используйте эту команду для отмены изменений, внесенных командой git add файлы. Вы также можете выполнить git reset чтобы восстановить все файлы на сцене.
- **git checkout <файлы>** — копирует файлы со сцены в рабочую директорию. Эту команду удобно использовать чтобы сбросить нежелательные изменения в рабочей директории.

Графически смысл этих действий можно изобразить следующим образом:

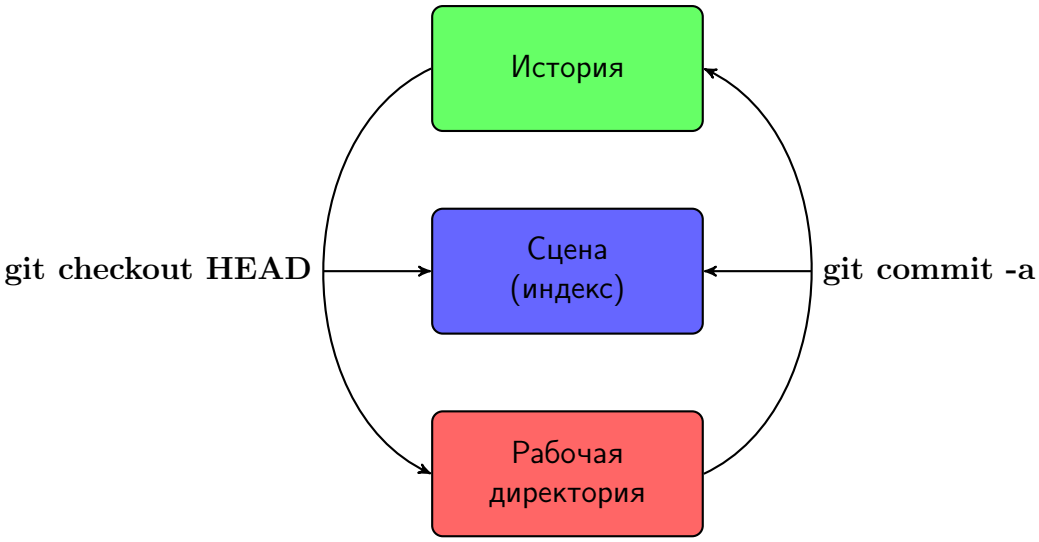


Вы можете использовать **git reset -p**, **git checkout -p**, и **git add -p** вместо (или вместе с) именами файлов, чтобы в интерактивном режиме выбирать, какие именно изменения будут скопированы.

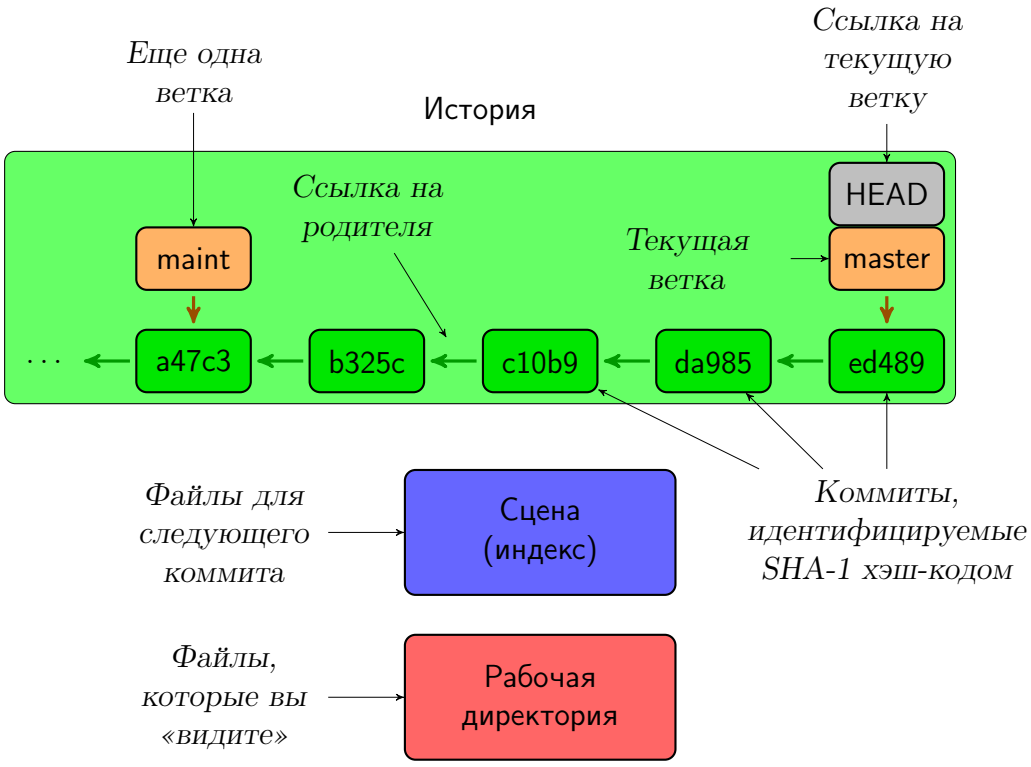
Также можно перепрыгнуть через сцену и сразу же получить файлы из истории прямо в рабочую директорию, или сделать коммит, минуя сцену.

- **git commit -a** — аналогичен запуску двух команд: **git add** для всех файлов, которые существовали в предыдущем коммите, и **git commit**.
- **git commit <файлы>** — создает новый коммит, в основе которого лежат уже существующие файлы, добавляя изменения только для указанных файлов. Одновременно, указанные файлы будут скопированы на сцену.
- **git checkout HEAD --<файлы>** — копирует файлы из текущего коммита и на сцену, и в рабочую директорию.

Схематически это выглядит так:

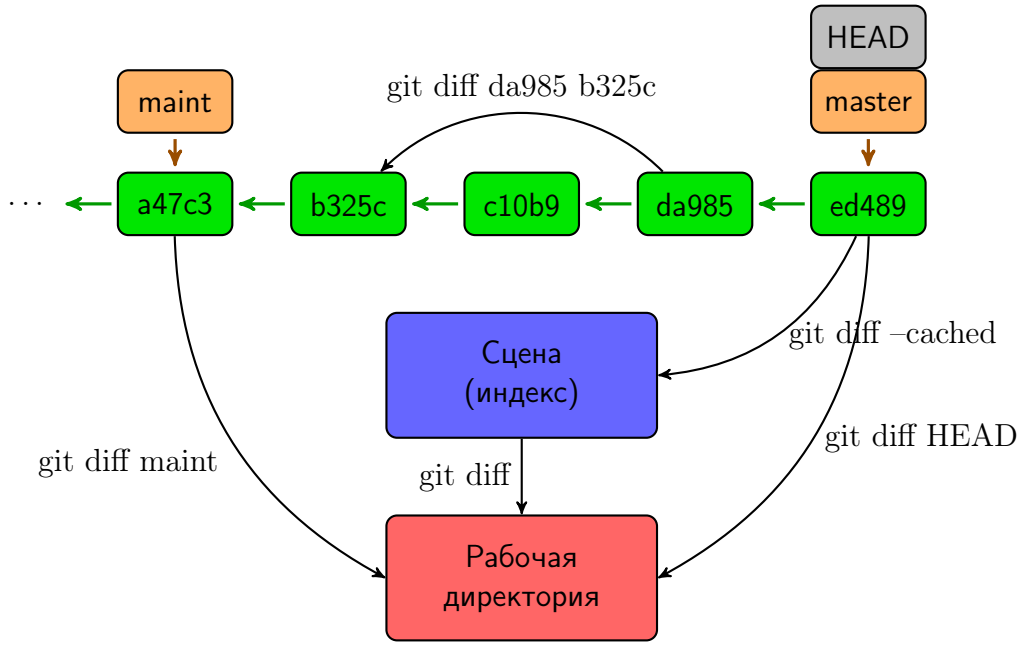


Чтобы двигаться дальше, изобразим схему внутреннего устройства истории, которую хранит Git. Типичная ситуация изображена на схеме ниже:



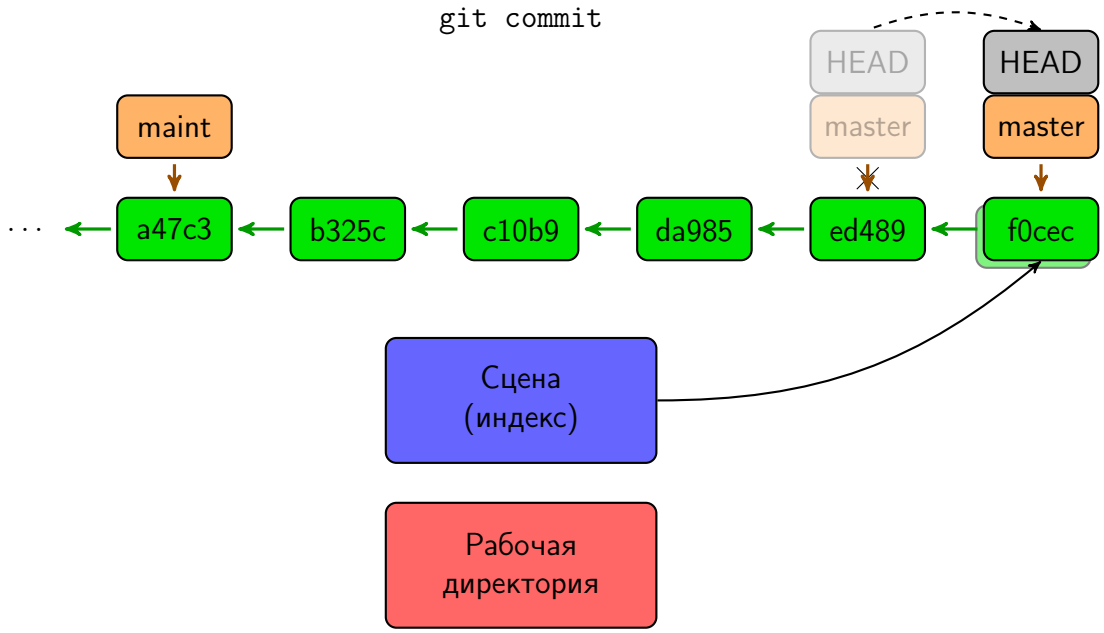
1.1 Команда diff

Как вы уже знаете из предыдущего материала, существует способ посмотреть, какие изменения были сделаны в разных коммитах и сравнить их между собой. Ниже на схеме изображены несколько простых примеров использования команды **git diff**. К каждой из этих команд можно добавить имена файлов в качестве дополнительного аргумента. При этом выводится информацию об изменениях только для перечисленных файлов.

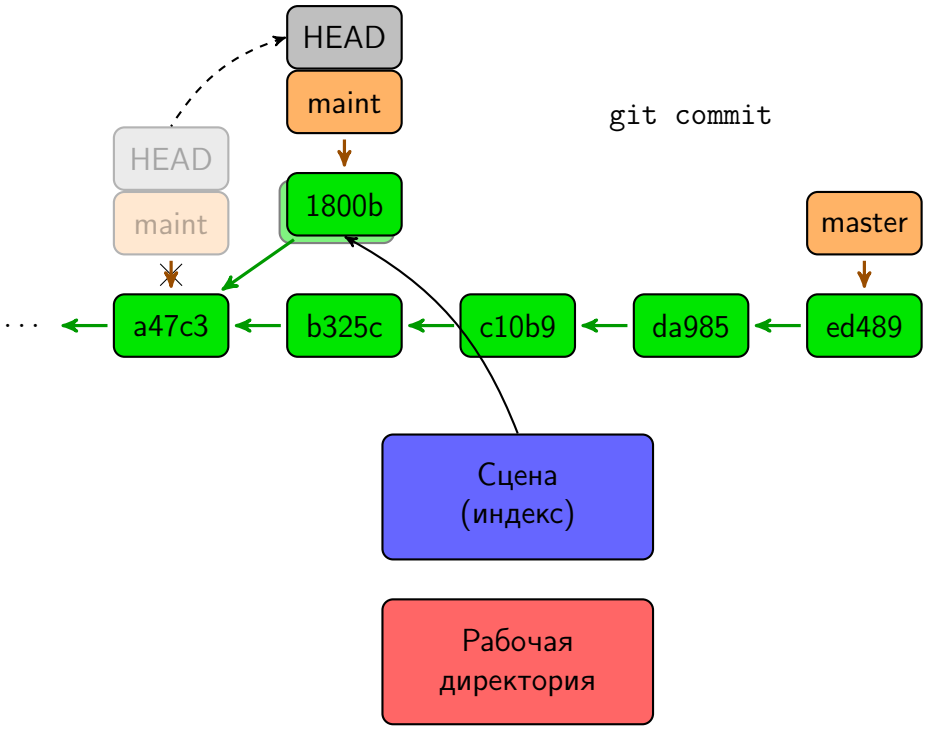


1.2 Команда commit

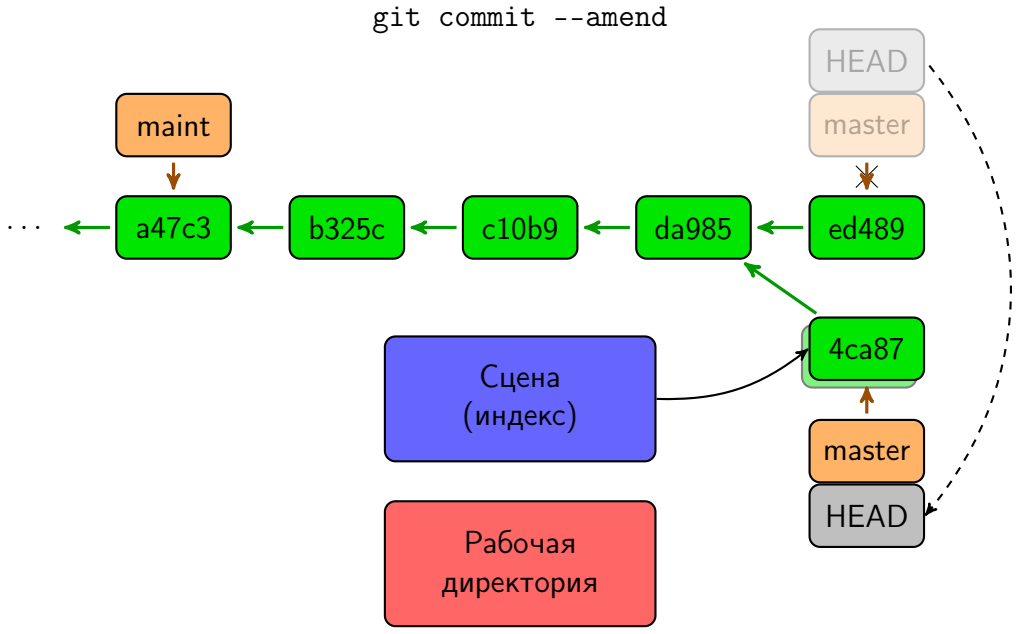
Когда вы делаете коммит, git создает новый объект коммита, используя файлы со сцены, а текущей коммит становится родителем для нового. После этого указатель текущей ветки перемещается на новый коммит. Вы это видите на картинке, где **master** — это текущая ветка. До совершения коммита **master** указывал на коммит **ed489**. После добавления нового коммита **f0cec**, родителем которого стал **ed489**, указатель ветки **master** был перемещен на новый коммит.



То же самое происходит, если одна ветка является предком другой ветки. Ниже показан пример нового коммита **1800b** в ветке **maint**, которая является предком ветки **master**. После этого ветка **maint** уже больше не является предком ветки **master**. И в случае необходимости объединения работы, проделанной в этих разделенных ветках, следует воспользоваться командой **merge** (что более предпочтительно) или **rebase**.



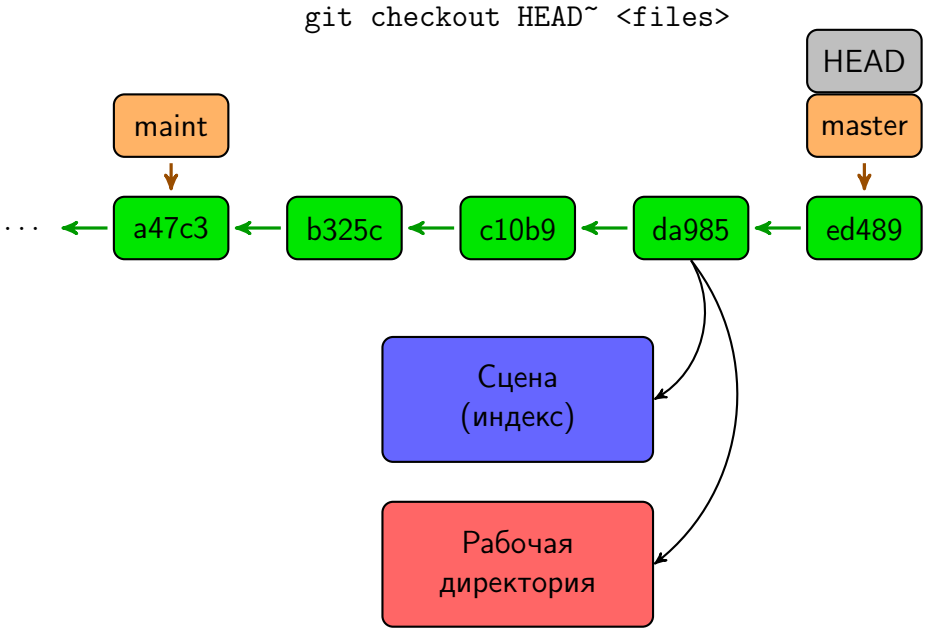
Наконец, если вы сделали ошибку в последнем коммите, её легко исправить с помощью команды **git commit --amend**. Эта команда создает новый коммит, родителем которого будет родитель ошибочного коммита. Старый ошибочный коммит будет отброшен, конечно же если только на него не будет ещё каких-либо других ссылок, что маловероятно.



1.3 Команда checkout

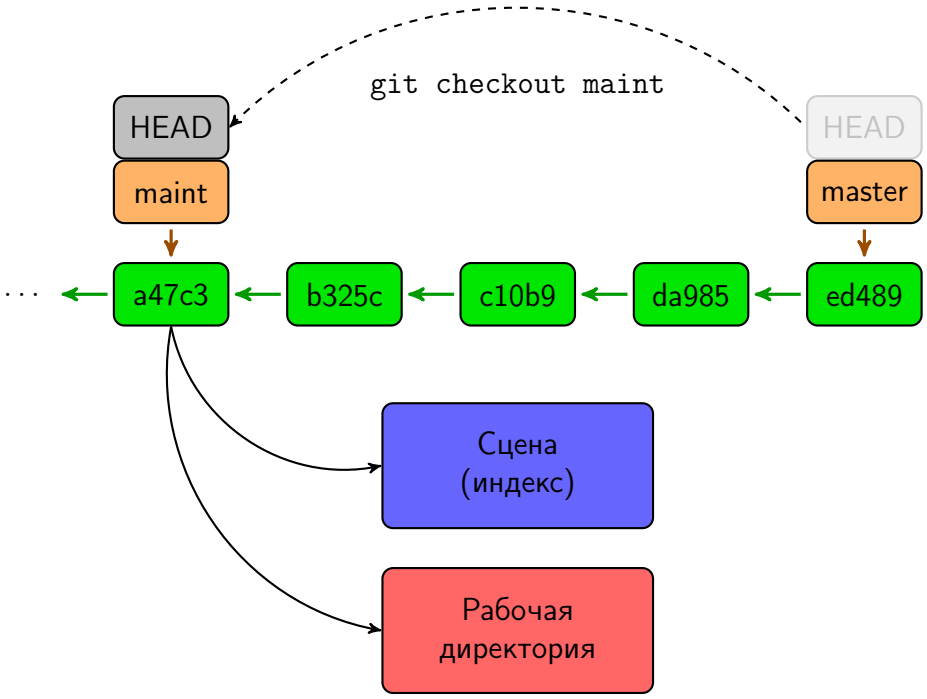
Команда **checkout** используется для копирования файлов из истории или сцены в рабочую директорию. Также она может использоваться для переключения между ветками.

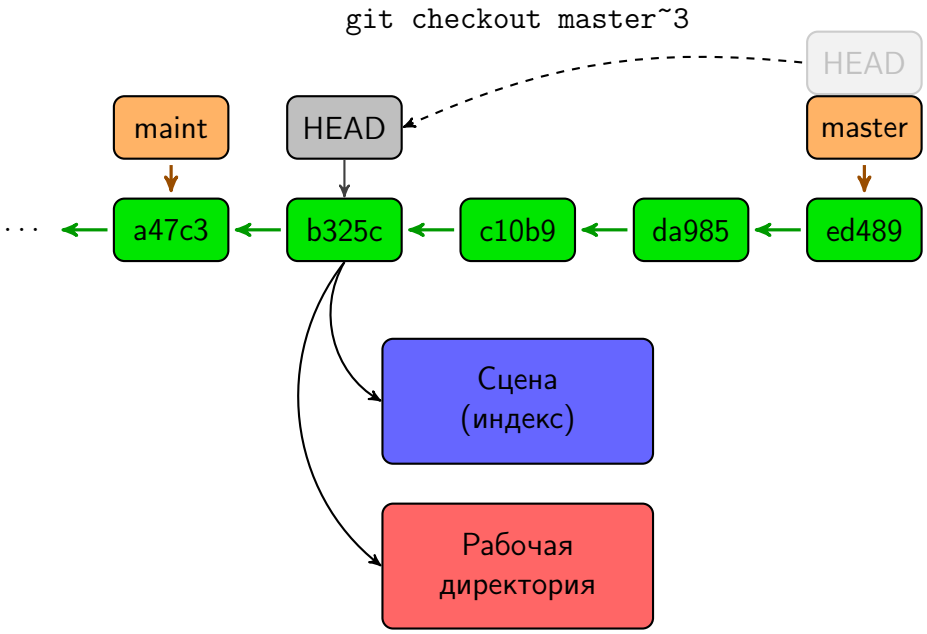
Когда вы указываете имя файла (и/или ключ **-p**), git копирует эти файлы из указанного коммита на сцену и в рабочую директорию. Например, **git checkout HEAD foo.cpp** копирует файл **foo.cpp** из коммита **HEAD** (предка текущего коммита) в рабочую директорию и на сцену. Если имя коммита не указано, то файл будет скопирован со сцены в рабочую директорию. Обратите внимание на то что при выполнении команды **checkout** позиция указателя текущей ветки **HEAD** остаётся прежней, указатель никуда не перемещается.



В том случае если мы не указываем имя файла, но указываем имя (локальной) ветки, то указатель **HEAD** будет перемещен на эту ветку (мы переключимся на эту ветку). При этом сцена и рабочая директория будут приведены в соответствие с этим коммитом. Любой файл, который присутствует в новом коммите (ниже на схеме это **a47c3**) будет скопирован из истории; любой файл, который был в старом коммите (**ed489**), но отсутствует в новом, будет удален; любой файл, который не записан ни в одном коммите, будет проигнорирован.

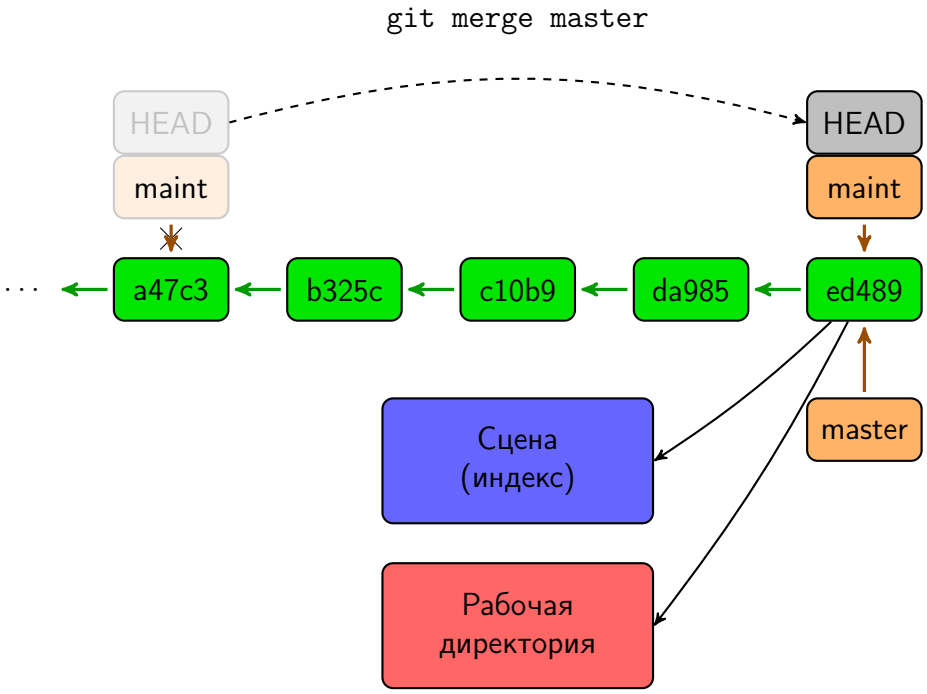
В том случае, если мы не указываем имя файла, и не указываем имя (локальной) ветки, а указываем тег, дистанционную (remote) ветку, SHA-1 хеш коммита или что-то вроде master 3, то мы получаем безымянную ветку, называемую «detached HEAD» («оторванная голова»). Это очень полезная штука для если нам надо осмотреться в истории коммитов, т.е. временно «окунуться» в прошлое внутри ветки и не связанное с определенным файлом.


git checkout maint
HEAD
maint
HEAD
master
...
a47c3
b325c
c10b9
da985
ed489
Сцена
(индекс)
Рабочая
директория

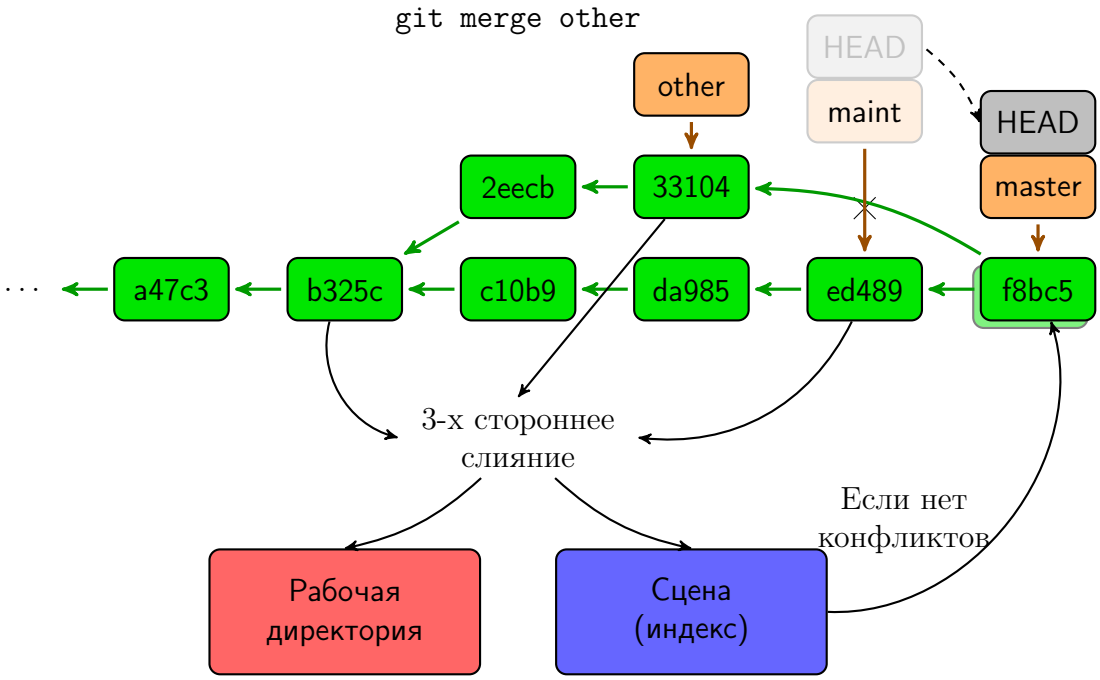

git checkout master~3
HEAD
maint
HEAD
master
...
a47c3
b325c
c10b9
da985
ed489
Сцена
(индекс)
Рабочая
директория

1.4 Команда merge

Команда **merge** (слияние) создает новый коммит на основе текущего коммита, применяя изменения других коммитов. Перед слиянием сцена должна быть приведена в соответствие с текущим коммитом. Самый простой случай слияния — это когда другой коммит является предком текущего коммита: в этом случае ничего не происходит. Другой простой случай слияния — когда текущий коммит является предком другого коммита: в этом случае происходит быстрая перемотка (fast-forward). Ссылка текущей ветки будет просто перемещена на новый коммит, а сцена и рабочая директория будут приведены в соответствие с новым коммитом.



Во всех других случаях выполняется «настоящее» слияние. Можно изменить стратегию слияния, но по умолчанию будет выполнено «рекурсивное» слияние, для которого будет взят текущий коммит (**ed489** ниже на схеме), другой коммит (**33104**) и их общий предок (**b325c**). Для этих трех коммитов будет выполнено трехстороннее слияние. Результат этого слияние будет записан в рабочую директорию и на сцену, а также будет добавлен результирующий коммит со вторым родителем (**33104**).


git merge other
other
HEAD
maint
HEAD
master
2eecb
33104
...
a47c3
b325c
c10b9
da985
ed489
f8bc5
3-х стороннее
слияние
Если нет
конфликтов
Рабочая
директория
Сцена
(индекс)