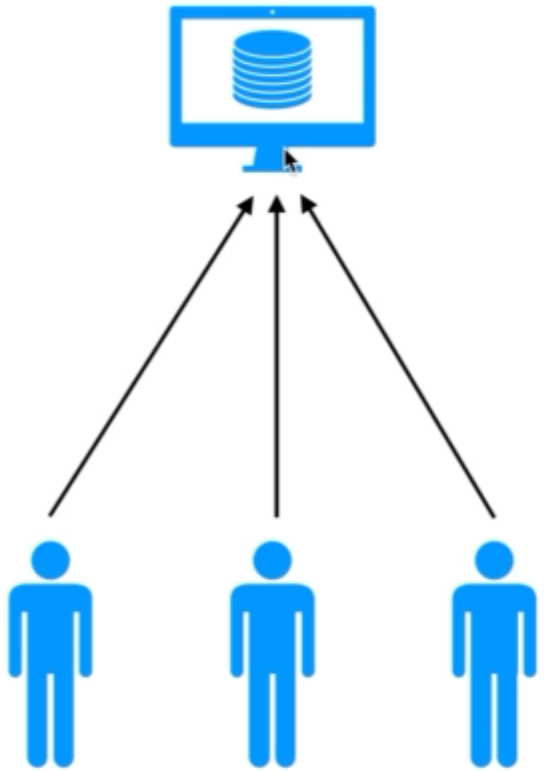
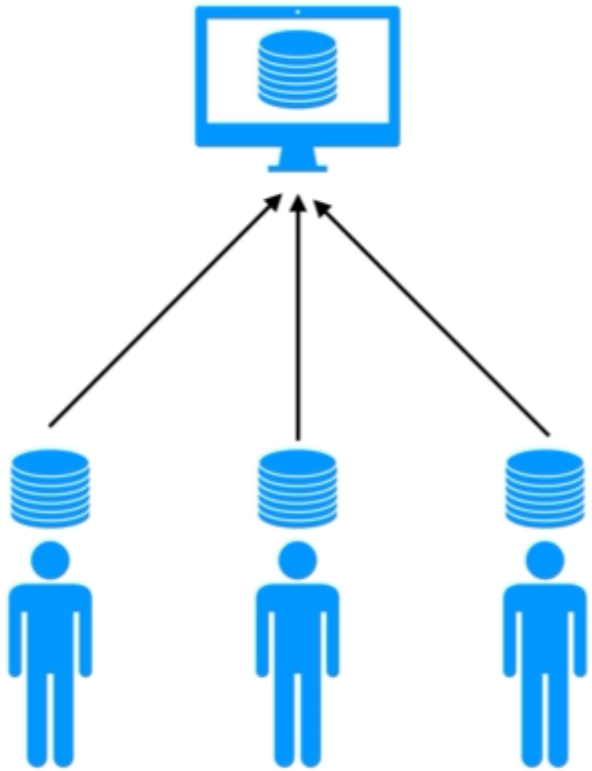
ASTON

Git

astondevs.ru

Централизованная и распределенная системы контроля версий

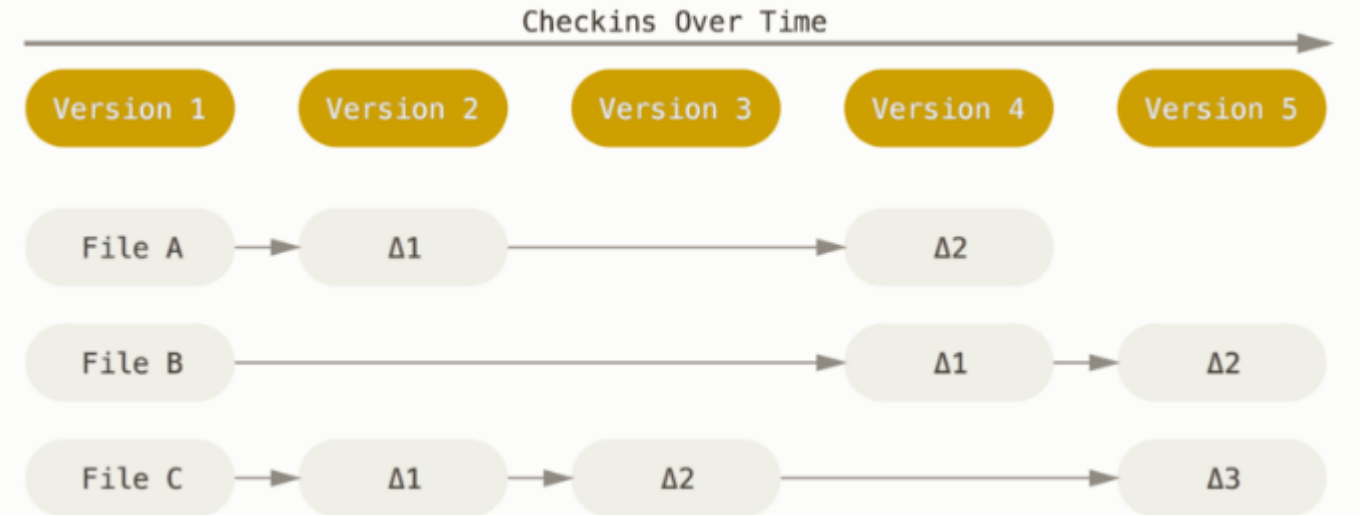




GIT

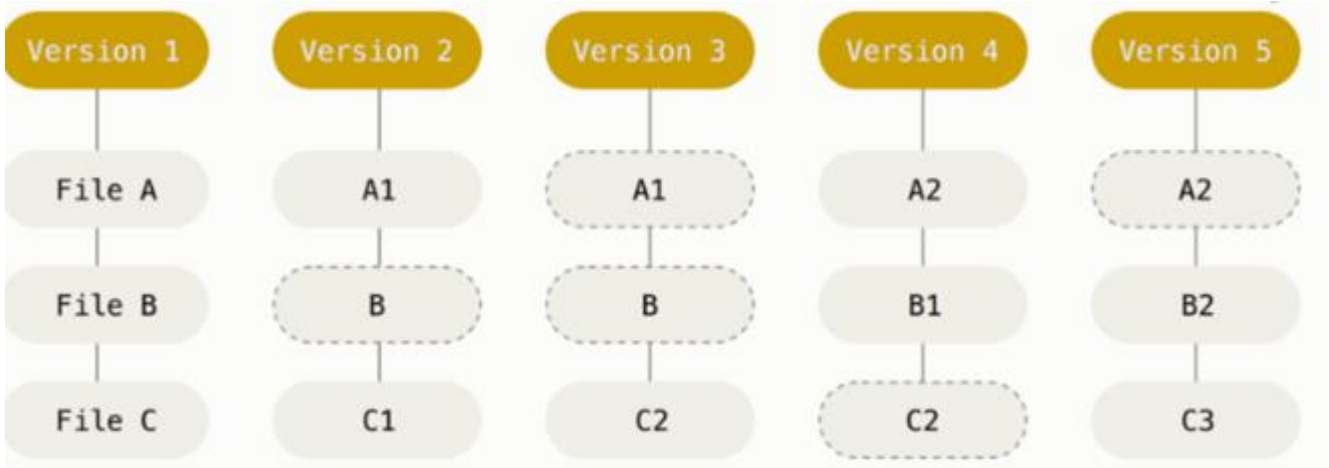
Снимки, а не различия

Основное отличие Git от любой другой СКВ (включая Subversion и её собратьев) — это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т.д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени (обычно это называют контролем версий, основанным на различиях).

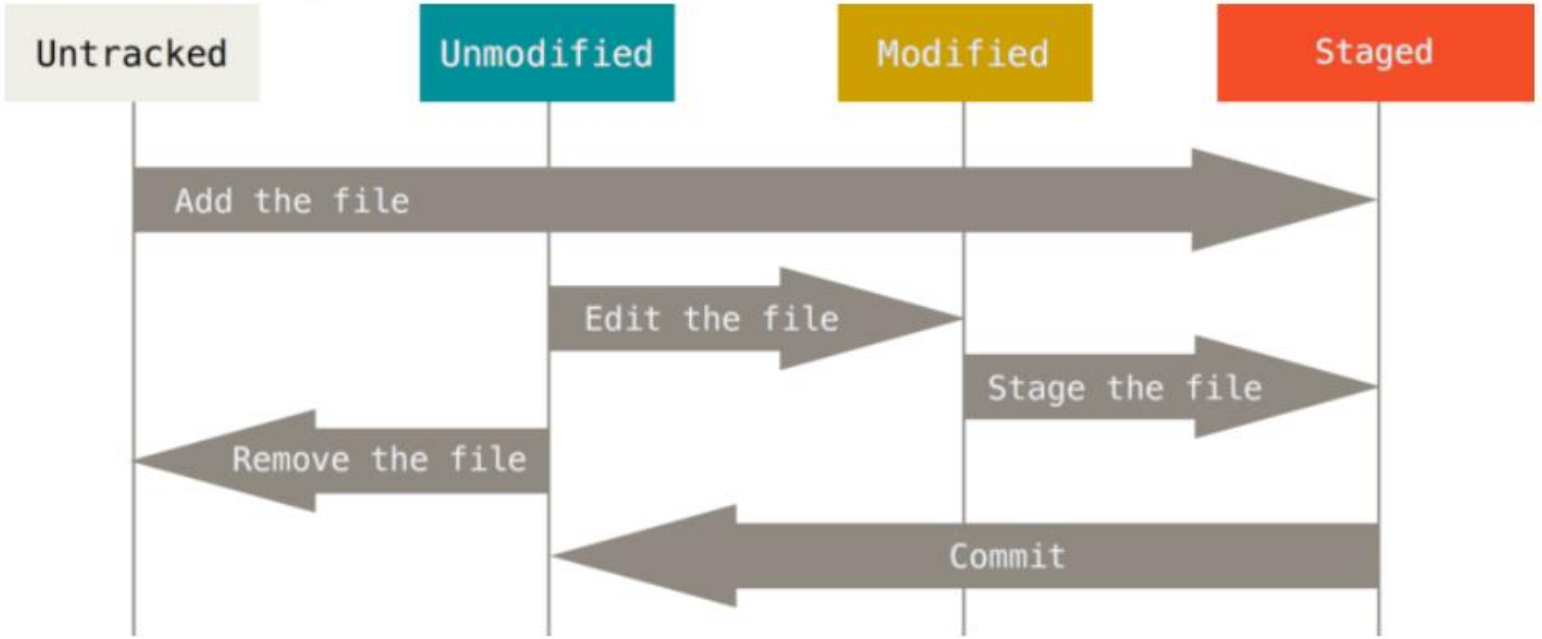


Снимки, а не различия

Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, поток снимков.

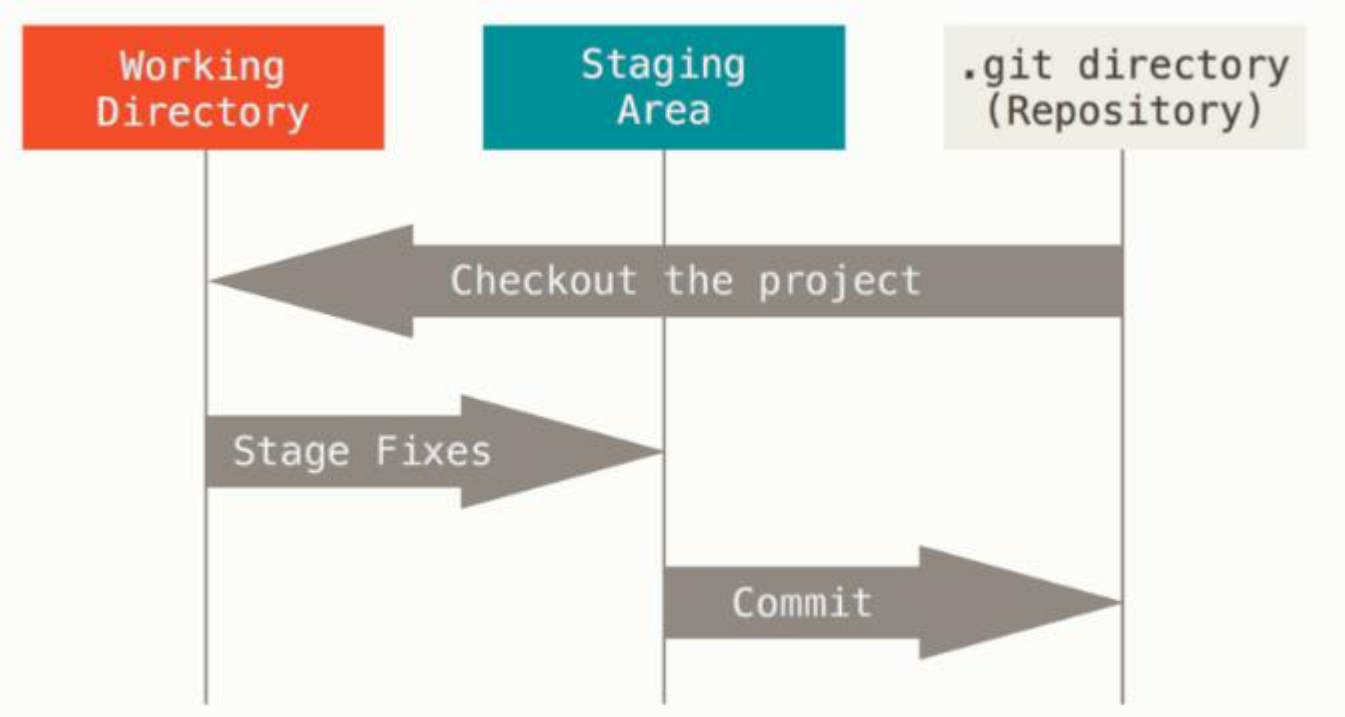


Жизненный цикл файлов



- **Untracked (неотлеживаемый)**: созданные файлы, про которые GIT ничего не знает. Перевести в состояние staged можно с помощью команды git add
- **Modified (измененный):** это файлы, которые были изменены с момента последней фиксации. Git отслеживает изменения в них, но они не были добавлены в индекс (Staging area или область подготовленных файлов). Вы можете добавить их в индекс с помощью команды git add.
- **Staged (подготовленный):** Это файлы, которые были изменены и добавлены в индекс с помощью git add, но еще не были зафиксированы с помощью git commit. Они готовы к фиксации в следующем коммите.
- **Committed (зафиксированный):** Это файлы, которые были зафиксированы в репозитории с помощью git commit. Они находятся в состоянии, которое можно считать "состоянием по умолчанию" в репозитории.

Три состояния



- **commited -** зафиксированный значит, что файл уже сохранён в вашей локальной базе.
- **modified -** к изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы.
- **staged -** подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.

Основные команды

git init – создание репозитория
git status – текущее состояние файлов в папке
git add – добавить отслеживания файла
git push – отправит изменения на репозиторий
git commit – сохранить изменения
git stash – сохраняет изменения в отдельную папку
git apply – возвращает изменения из stash
git reset – используется для отмены изменений (три режима SOFT, MIXED, HARD
git revert – используется для отмены изменений, но создается новый коммит
git ammend – добавление изменений в последний коммит
git squash – объединение двух коммитов
git pull - используется для извлечения и загрузки содержимого из удаленного репозитория
git rebase [-i | --interactive] – используется для перезаписи истории посредством изменения самих коммитов, а также информации в них

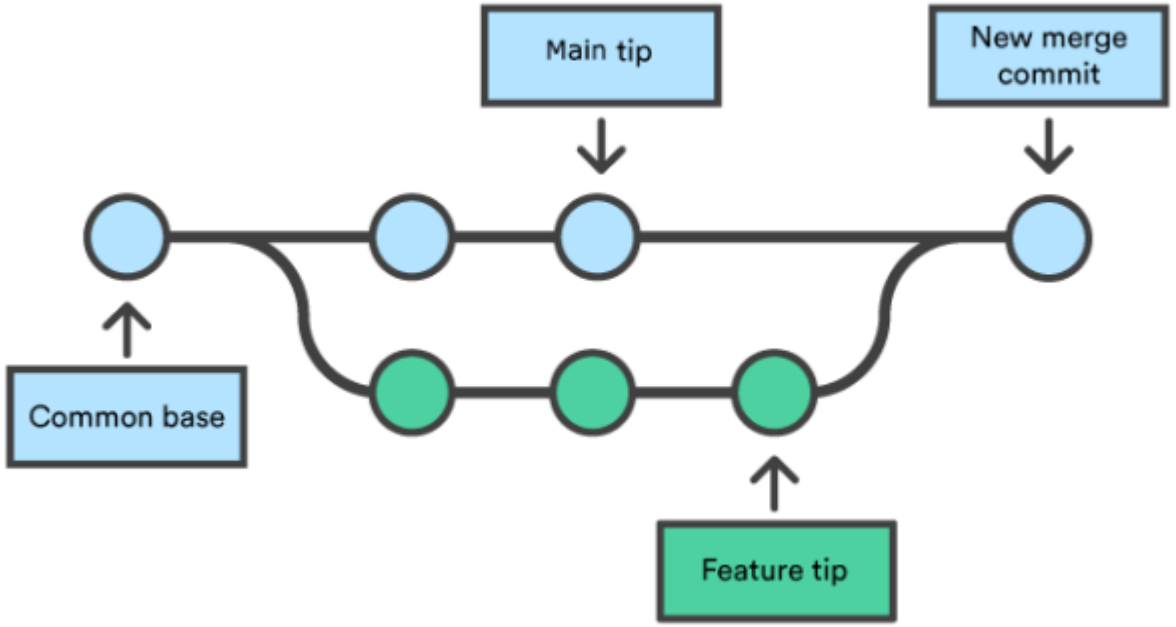
.gitignore Java

```
1  ###############################
2  ## Java
3  ###############################
4  .mtj.tmp/
5  *.class
6  *.jar
7  *.war
8  *.ear
9  *.nar
10 hs_err_pid*
```

.gitignore – файл который показывает гиту какие типы файлов игнорировать

Git merge

Слияние — обычная практика для разработчиков, использующих системы контроля версий. Независимо от того, созданы ли ветки для тестирования, исправления ошибок или по другим причинам, слияние фиксирует изменения в другом месте. Слияние принимает содержимое ветки источника и объединяет их с целевой веткой. В этом процессе изменяется только целевая ветка. История исходных веток остается неизменной.



Плюсы:

- простота;
- сохраняет полную историю и хронологический порядок;
- поддерживает контекст ветки.

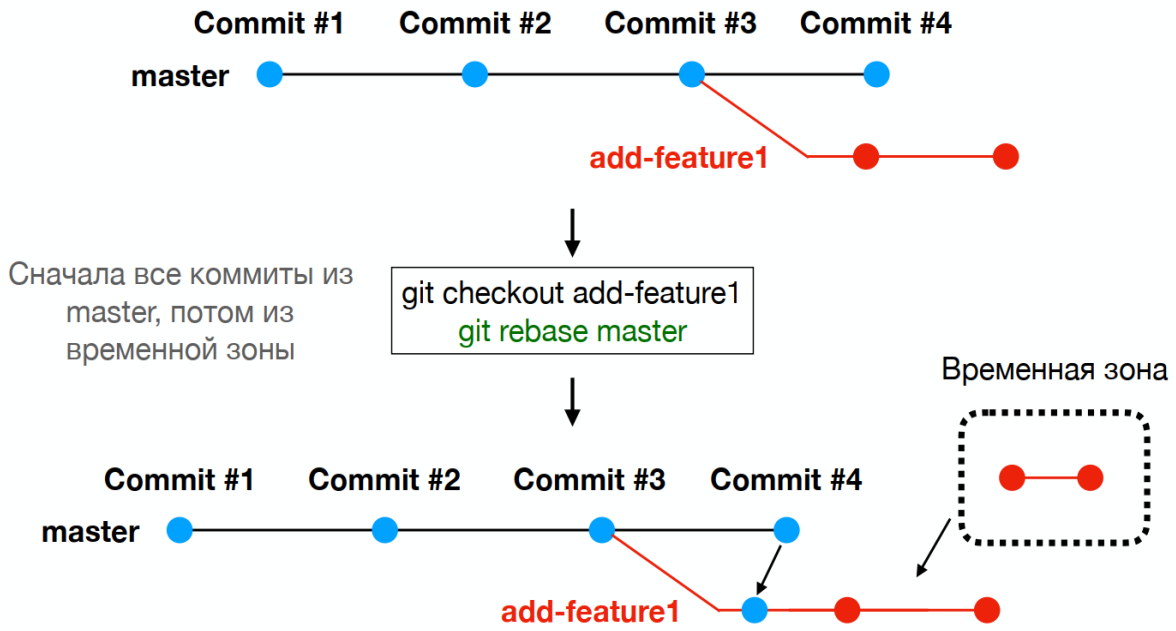
Минусы:

- история коммитов может быть заполнена (загрязнена) множеством коммитов;
- отладка с использованием git bisect может стать сложнее.

Git rebase

Перемещение — еще один способ перенести изменения из одной ветки в другую. Rebase сжимает все изменения в один «патч». Затем он интегрирует патч в целевую ветку.

В отличие от слияния, перемещение перезаписывает историю, потому что она передает завершенную работу из одной ветки в другую. В процессе устраняется нежелательная история.



Плюсы:

- Упрощает потенциально сложную историю
- Упрощение манипуляций с единственным коммитом
- Избежание слияния коммитов в занятых репозиториях и ветках
- Очищает промежуточные коммиты, делая их одним коммитом, что полезно для DevOps команд

Минусы:

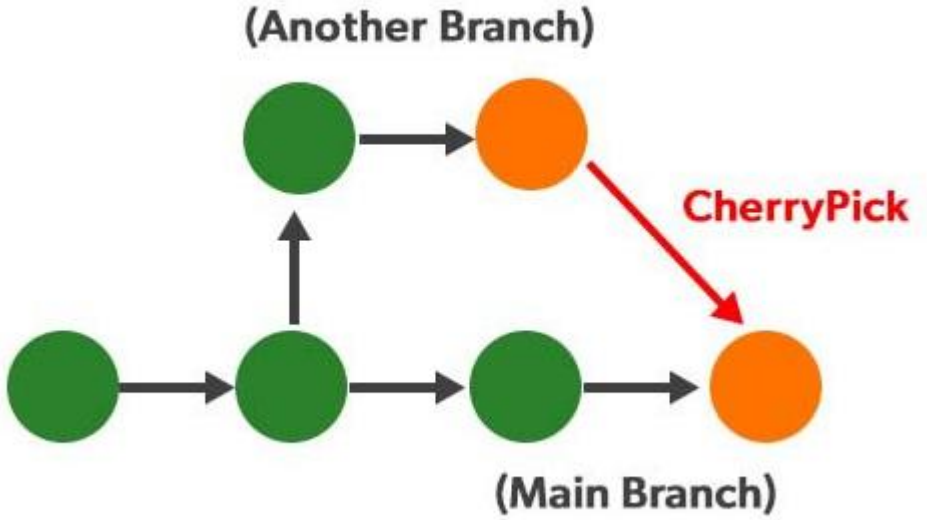
- Сжатие фич до нескольких коммитов может скрыть контекст
- Перемещение публичных репозиториев может быть опасным при работе в команде
- Появляется больше работы
- Для восстановления с удаленными ветками требуется принудительный пуш. Это приводит к обновлению всех веток, имеющих одно и то же имя, как локально, так и удаленно, и это ужасно.

Git Cherry Pick

git cherry-pick - это команда в системе контроля версий Git, которая позволяет выбирать и применять (переносить) коммиты из одной ветки в другую. Эта команда полезна, когда вы хотите применить только определенные коммиты из одной ветки в другую, а не весь набор изменений.

```
git cherry-pick <commit-hash>
```

Здесь <commit-hash> - это хеш-идентификатор коммита, который вы хотите применить. Вы можете указать один или несколько хешей коммитов для применения.
Когда вы выполняете git cherry-pick, Git применяет указанные коммиты в текущую ветку, создавая новые коммиты с аналогичными изменениями. Таким образом, командой git cherry-pick вы переносите выбранные коммиты из одной ветки в другую.
Важно отметить, что при применении коммитов с помощью git cherry-pick в другую ветку, вы можете столкнуться с конфликтами слияния, особенно если применяемые коммиты касаются тех же файлов или строк кода, которые уже были изменены в целевой ветке. В таких случаях вам придется разрешить конфликты вручную перед фиксацией изменений.



Литература

- https://git-scm.com/
- https://github.com/progit/progit2-ru/releases/download/2.1.71/progit_v2.1.71.pdf - книга
- https://www.youtube.com/watch?v=DK2PsTcSFFM&list=PLg5SS_4L6LYstwxTEOU05E0URTHnbtA0l&index=1 – курс
- https://www.youtube.com/watch?v=4-NX17lp-xQ&list=PLmRNNqEA7JoM77hOJkPrLOfJQGizCLR3P – курс (опционально)