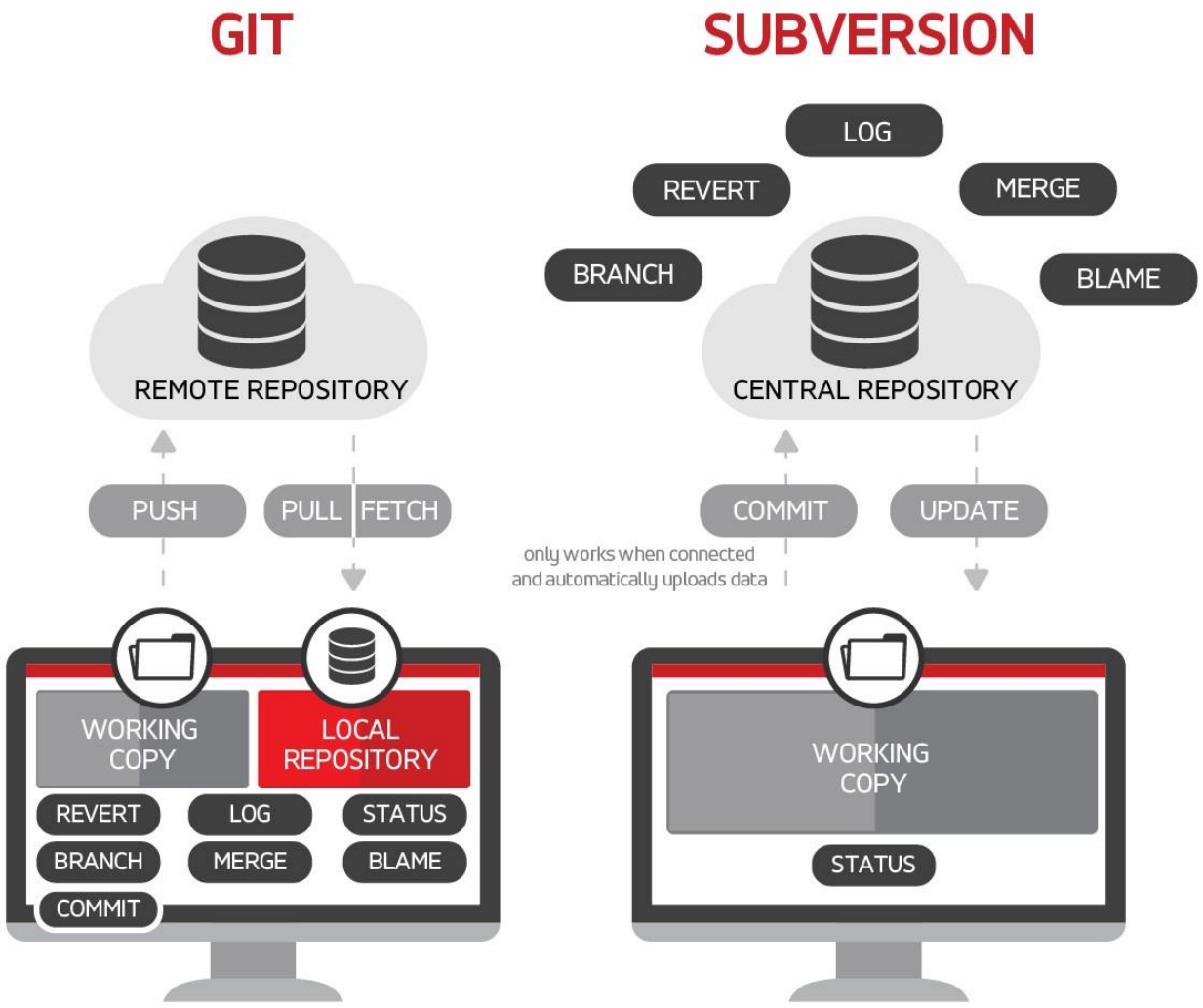
Git Introduction

Зачем нужен VCS?

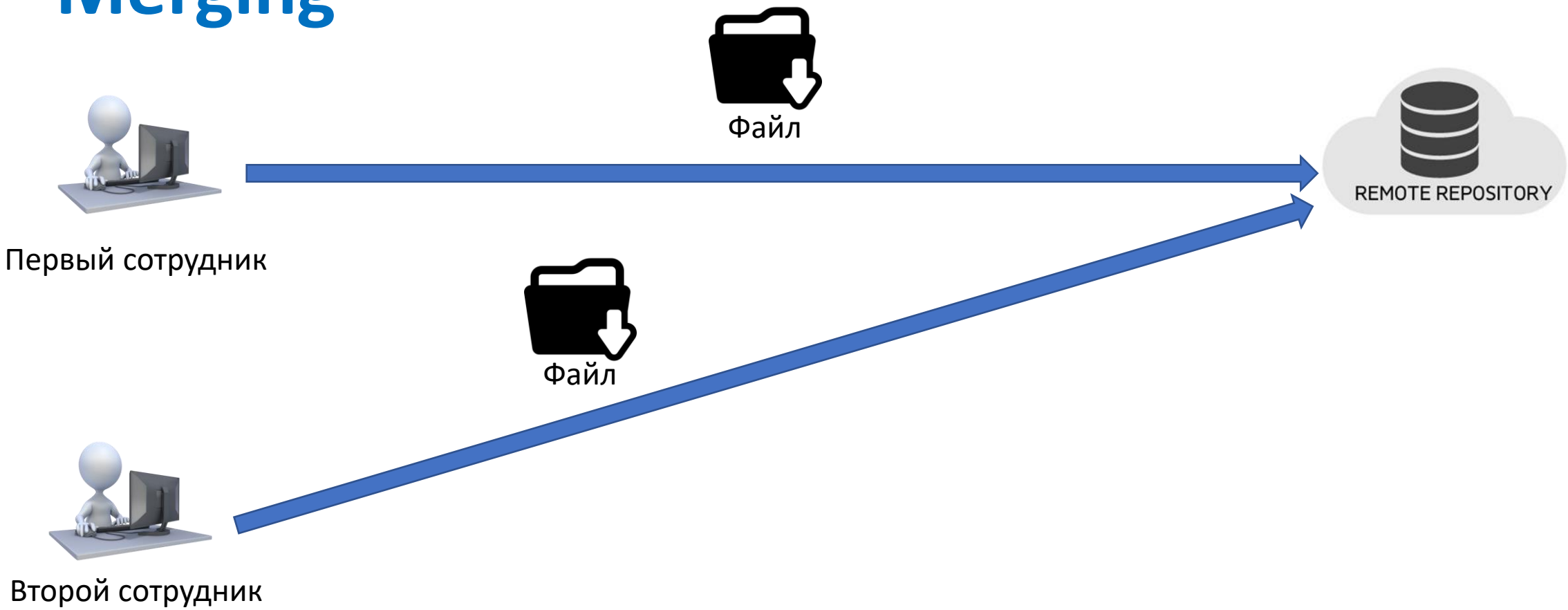
Сотрудник

1. Обеспечение ежедневного цикла (обновление->модификация->фиксация изменений)
2. Слияние версий
3. Разрешение конфликтов
4. Версионирование
5. Блокировки
6. Ветвление

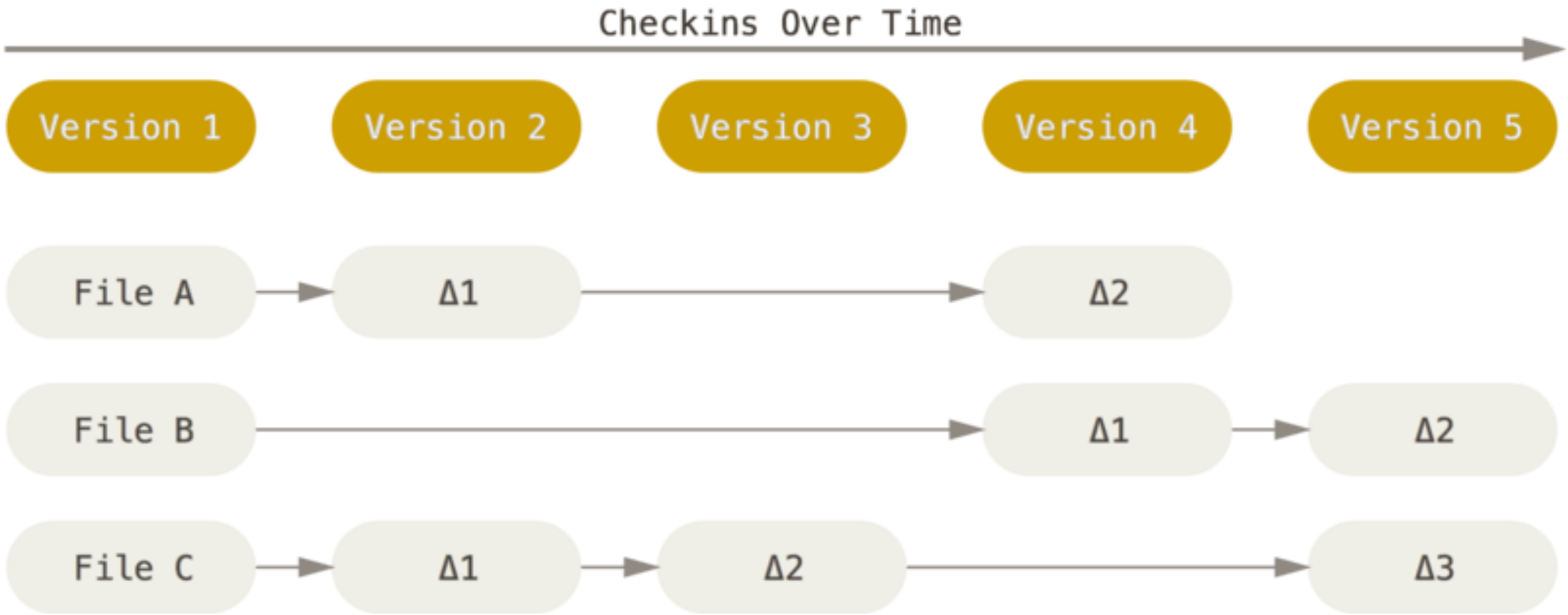
Git vs Subversion



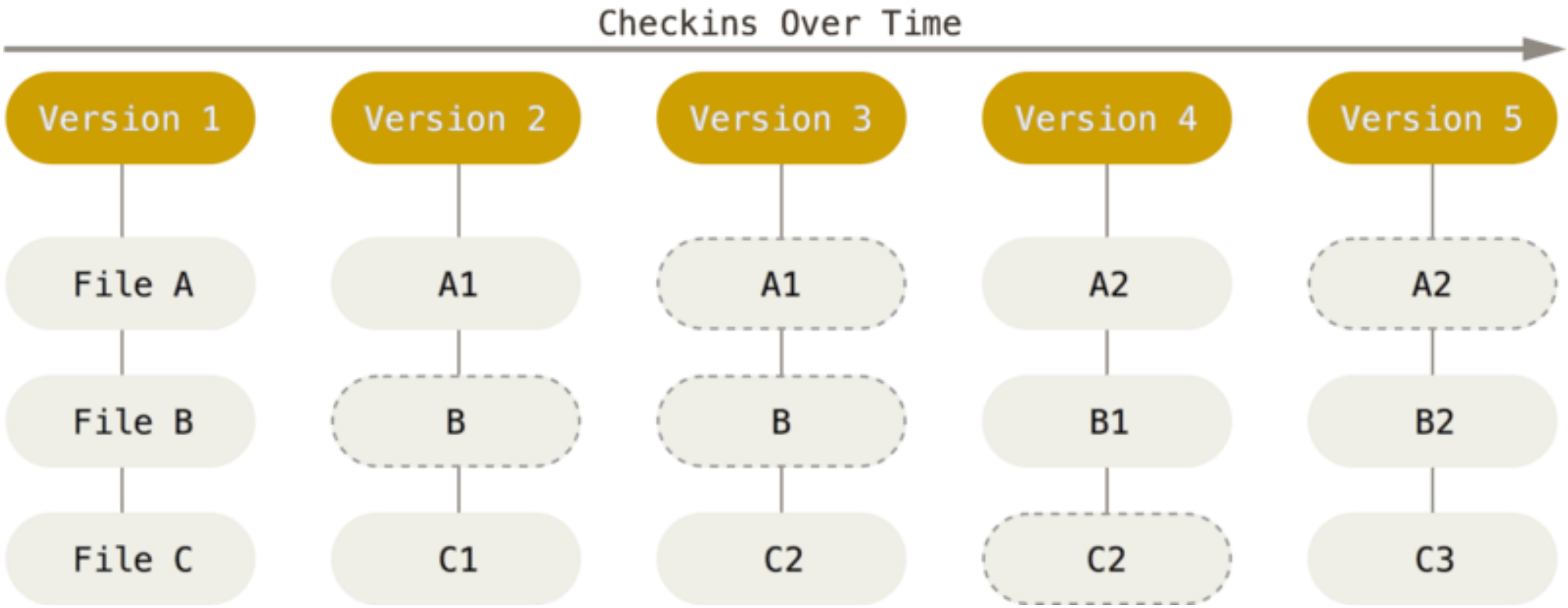
GIT обеспечивает слияние версий - Merging



CVS, Subversion, Perforce, Bazaar, and so on *delta-based* version control



GIT - *stream of snapshots*



GIT – установка

https://git-scm.com/book/ru/v2 - документация
https://git-scm.com/book/ru/v2/Введение-Установка-Git - инструкция по установке

Инсталляция MAC

- Открываем терминал
- Вводим команду git –version. Если git не установлен, MAC предложит это сделать

Инсталляция Linux

- Открываем терминал
- RHEL/CentOs выполнить команду sudo **dnf** install **git-all**
- DEBIAN/Ubuntu выполнить команду sudo **apt** install **git**

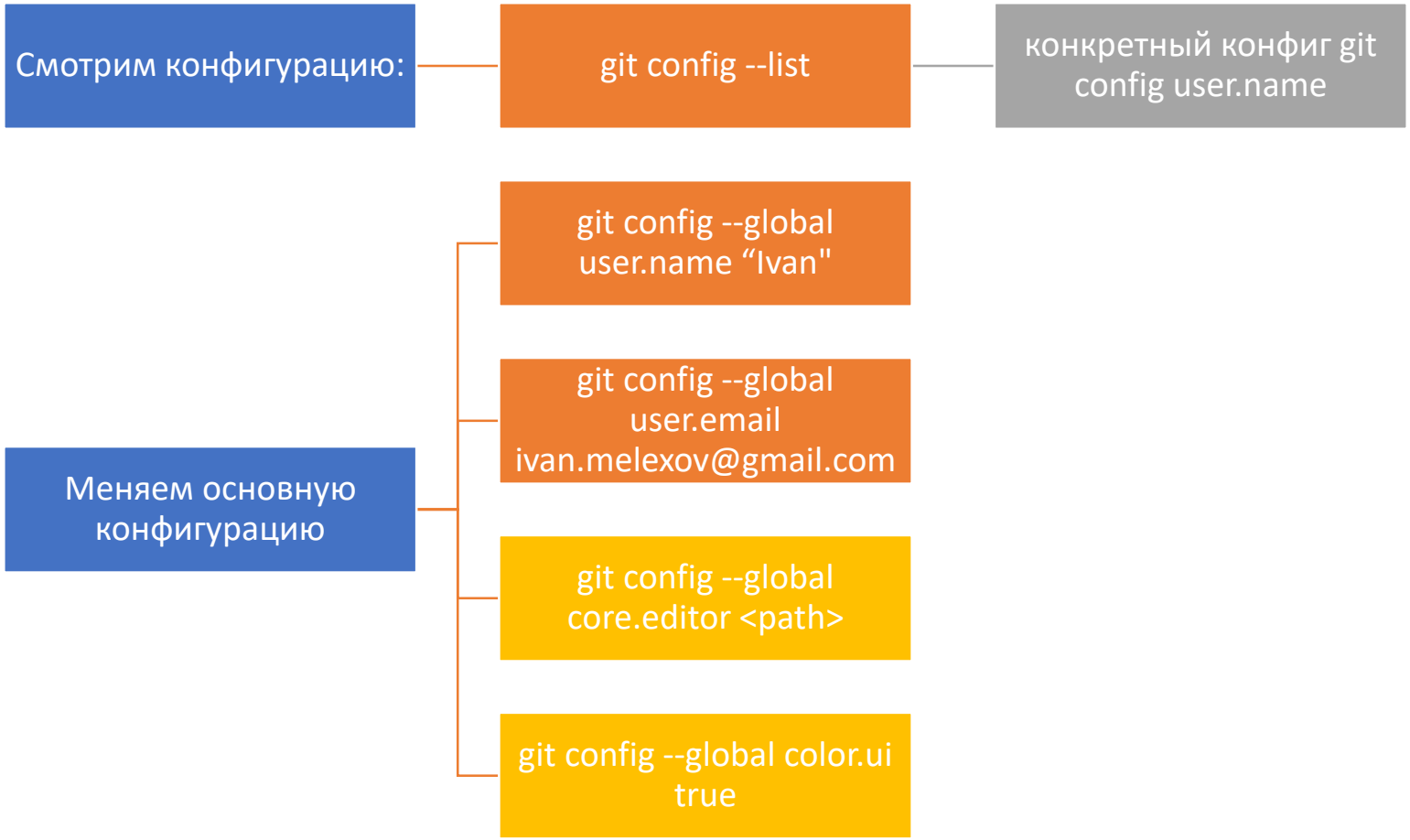
Инсталляция Windows

- Посетить сайт https://git-scm.com/download/win и следовать инструкциям инсталлятора

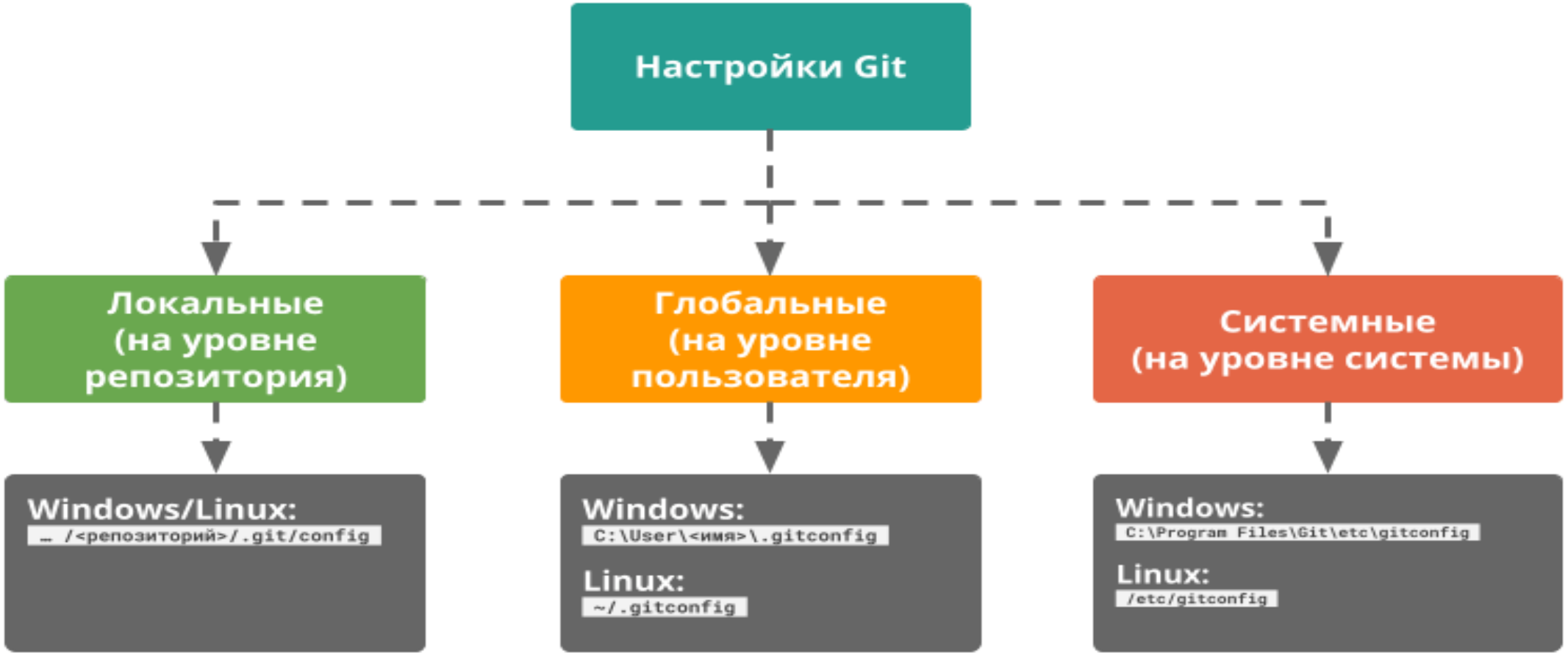
Базовые команды:

- git help
- git help <command name> (например git help merge)
- git version

GIT – первичная настройка



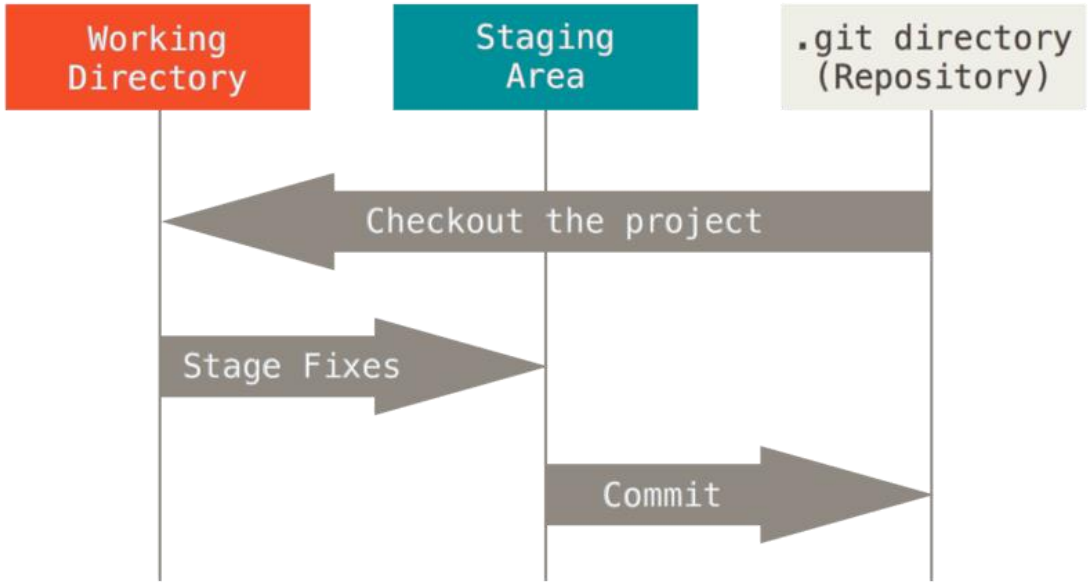
GIT – первичная настройка



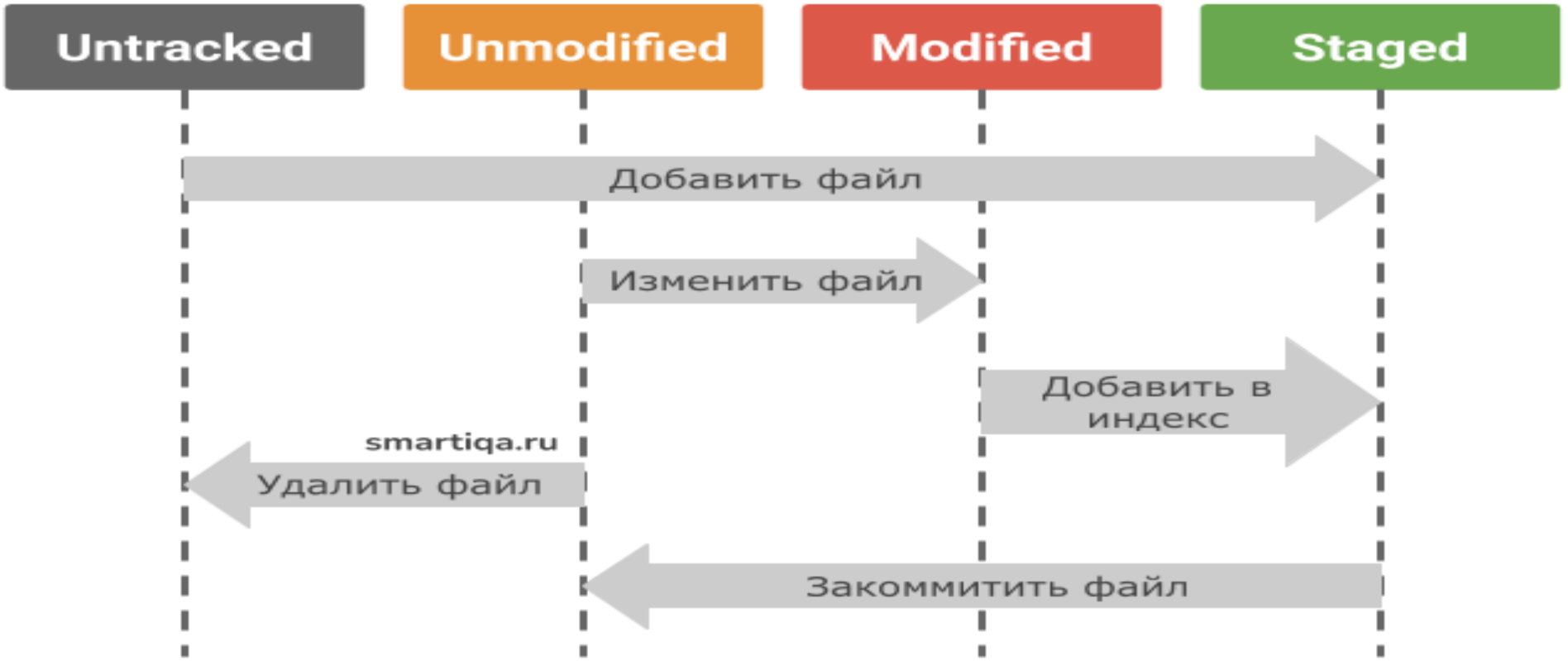
Три ключевых области

Действие	Результат
Создание файла	**Статус “untracked”**
Изменение файла	**Статус “modified”**
git add	**Статус “staged” (подготовленный)**
git commit	**Статус “committed” (зафиксированный)**

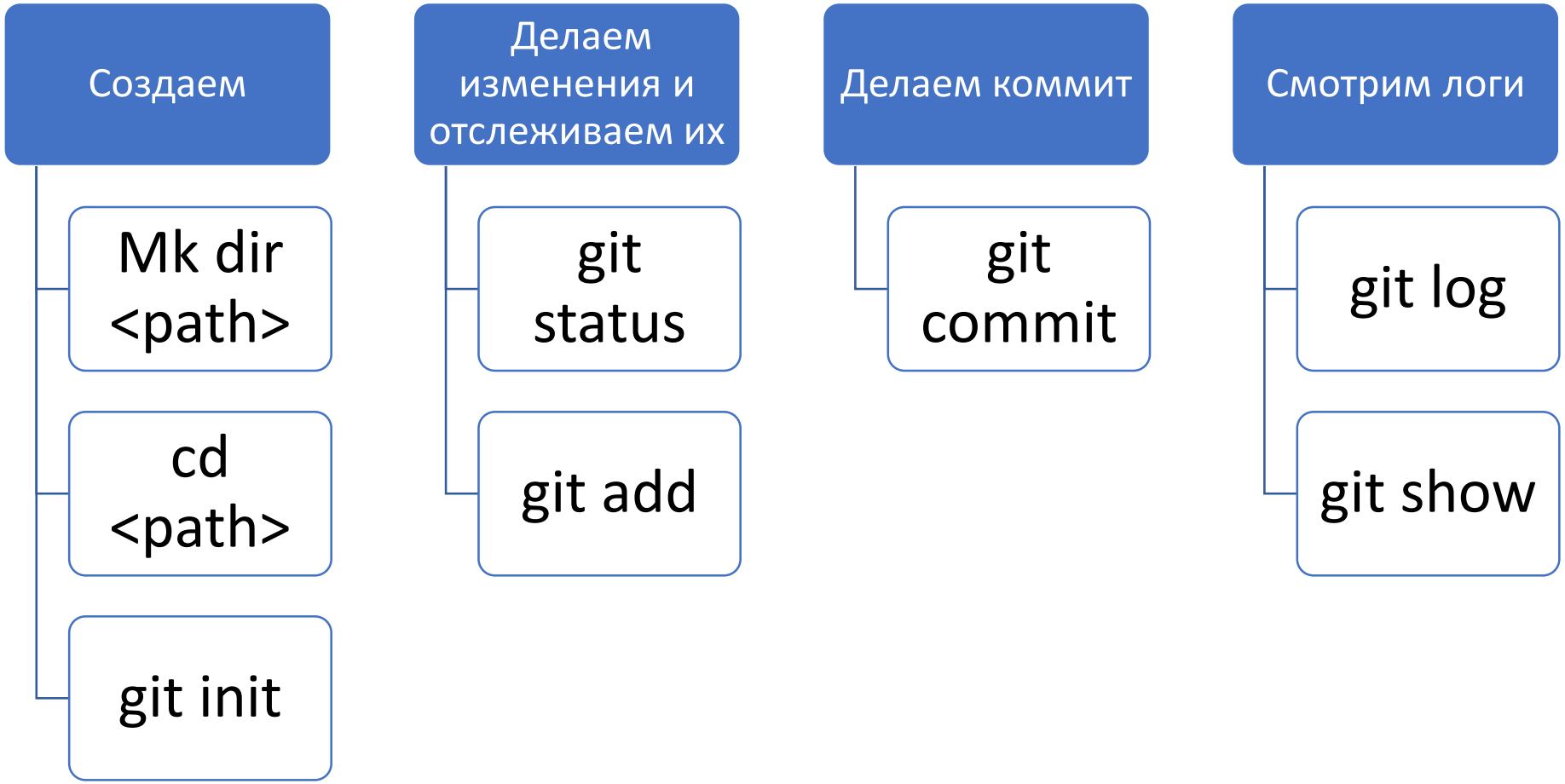
Modified - means that you have changed the file but have not committed it to your database yet
Staged - means that you have marked a modified file in its current version to go into your next commit snapshot.
Committed - means that the data is safely stored in your local database



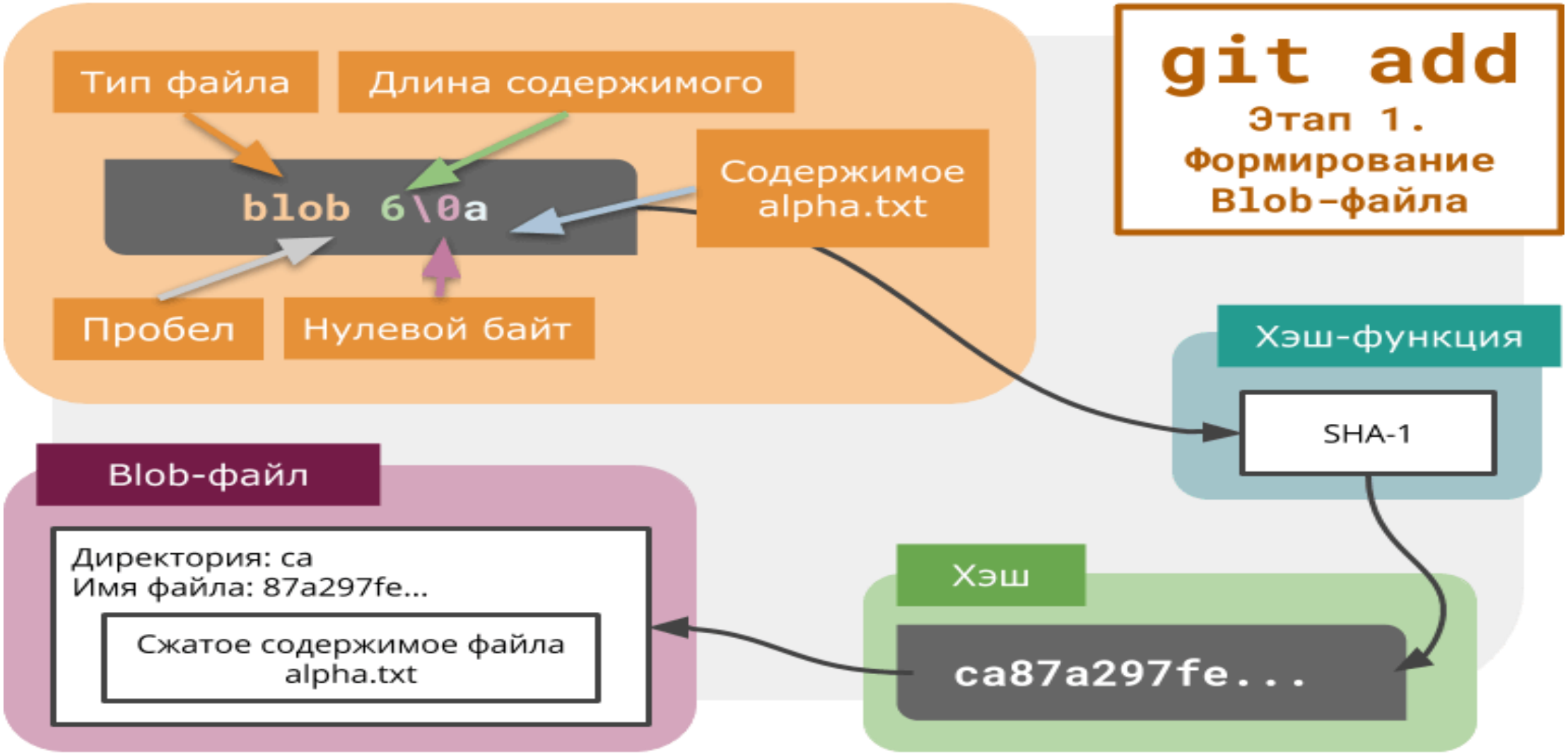
GIT – состояния файлов



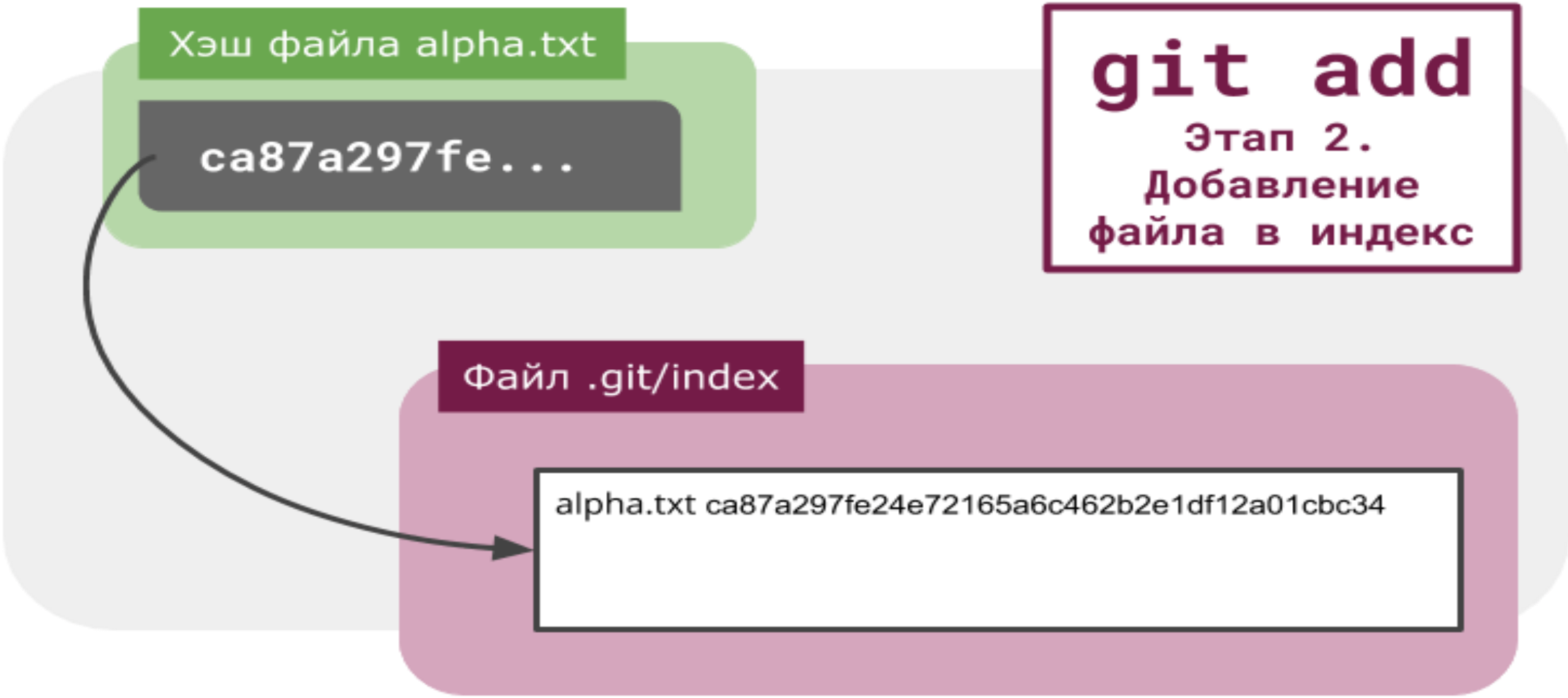
GIT - создаем первый проект



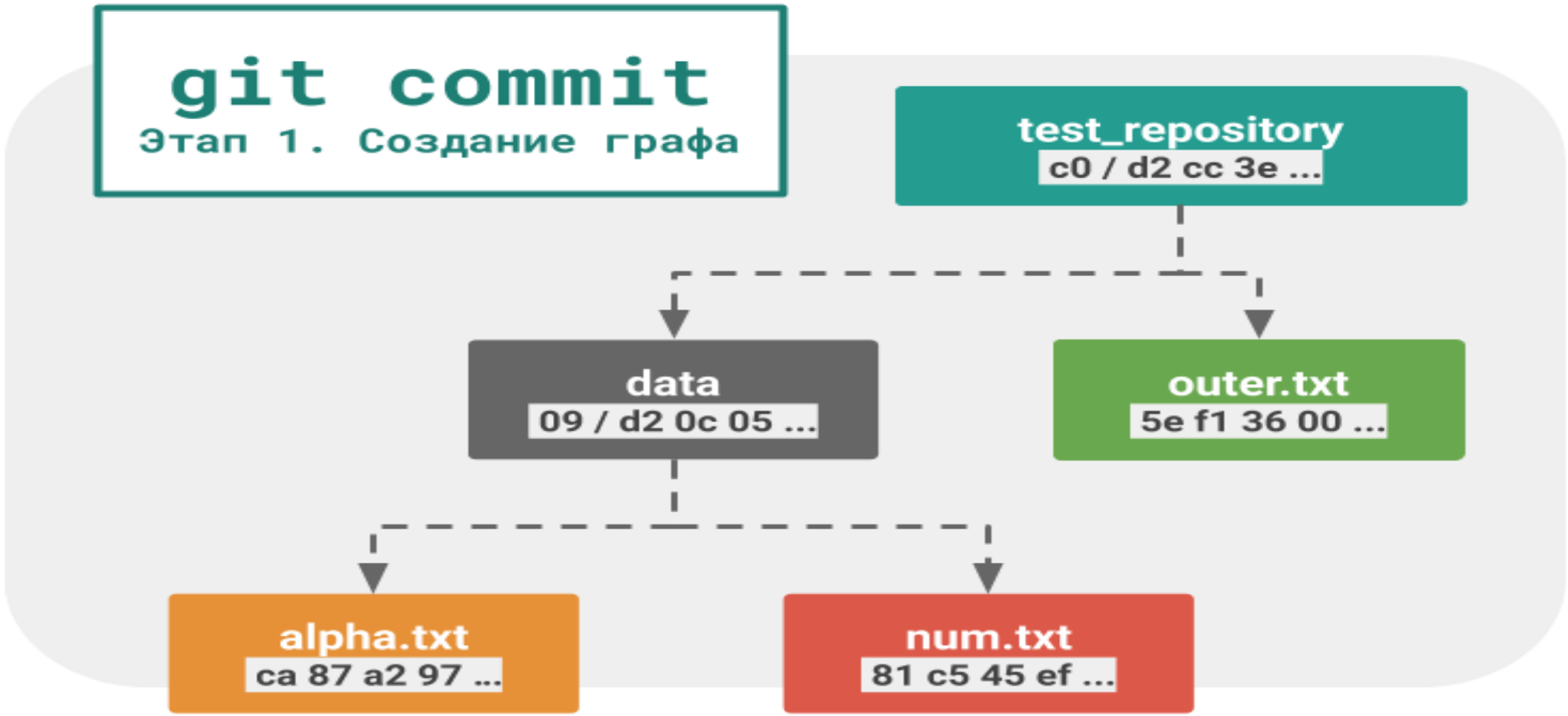
GIT add – что произошло (1 этап)



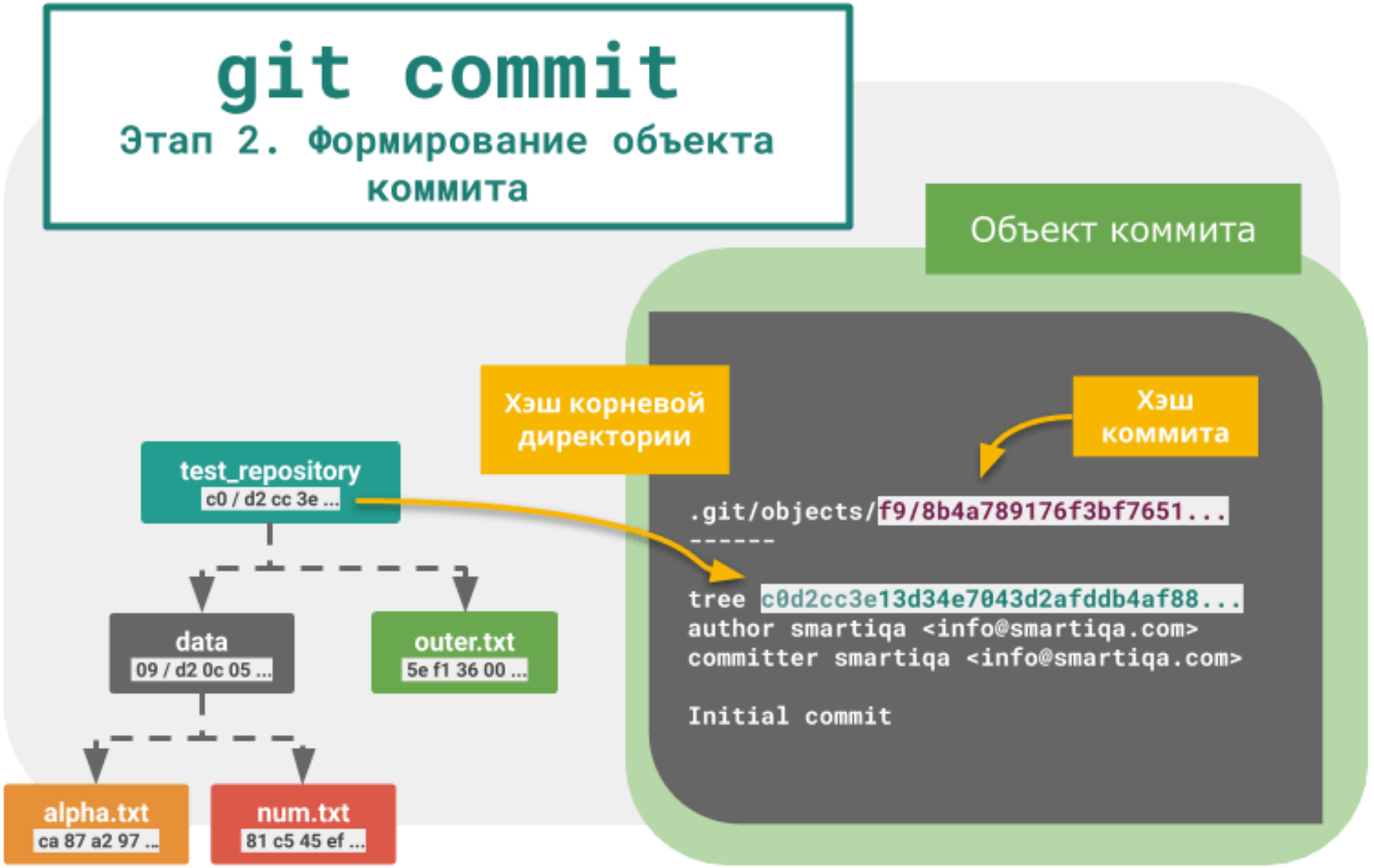
GIT add – что произошло (2 этап)



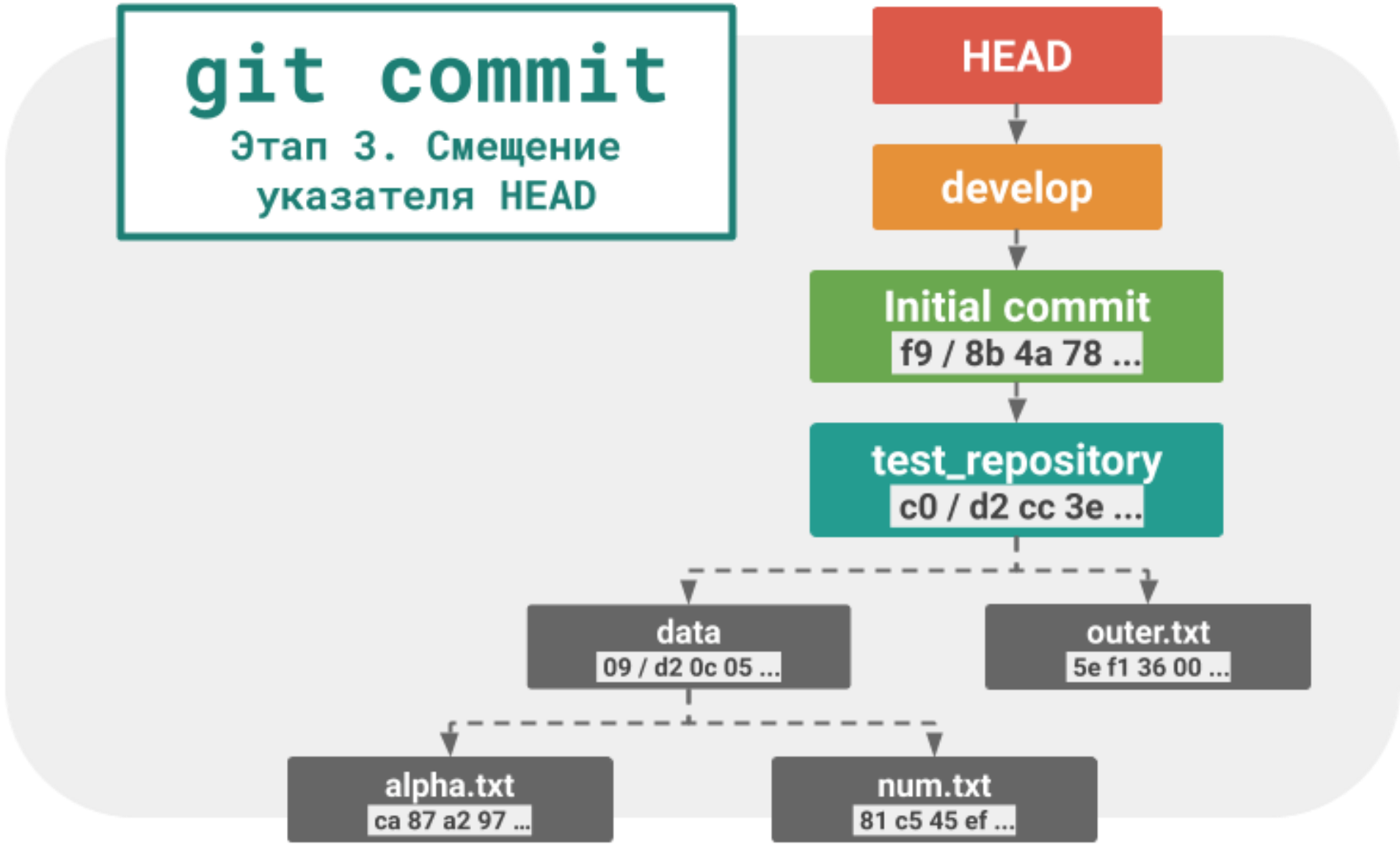
GIT commit – Этап 1. Создание графа



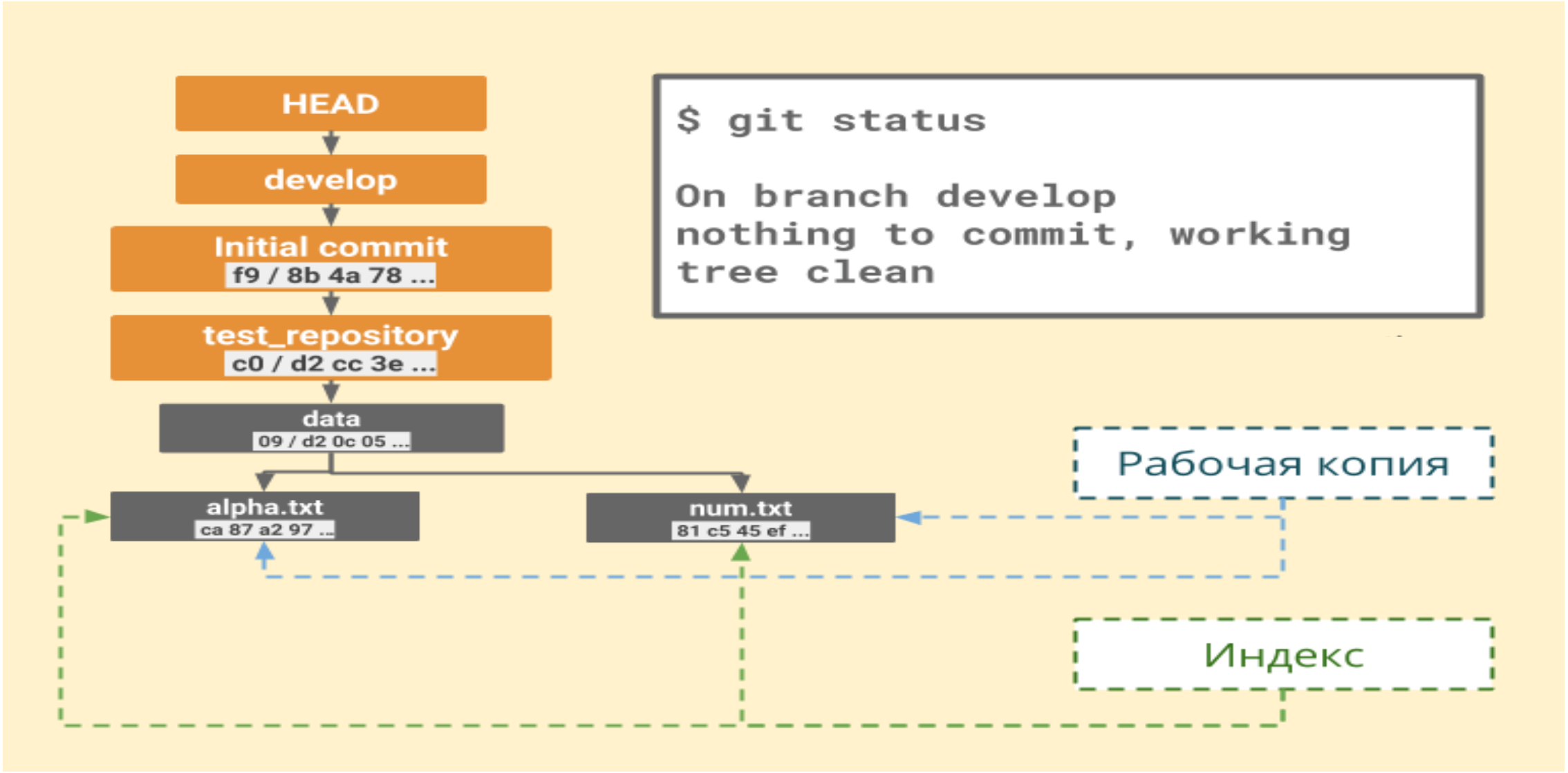
GIT commit – Этап 2. Создание объекта коммита



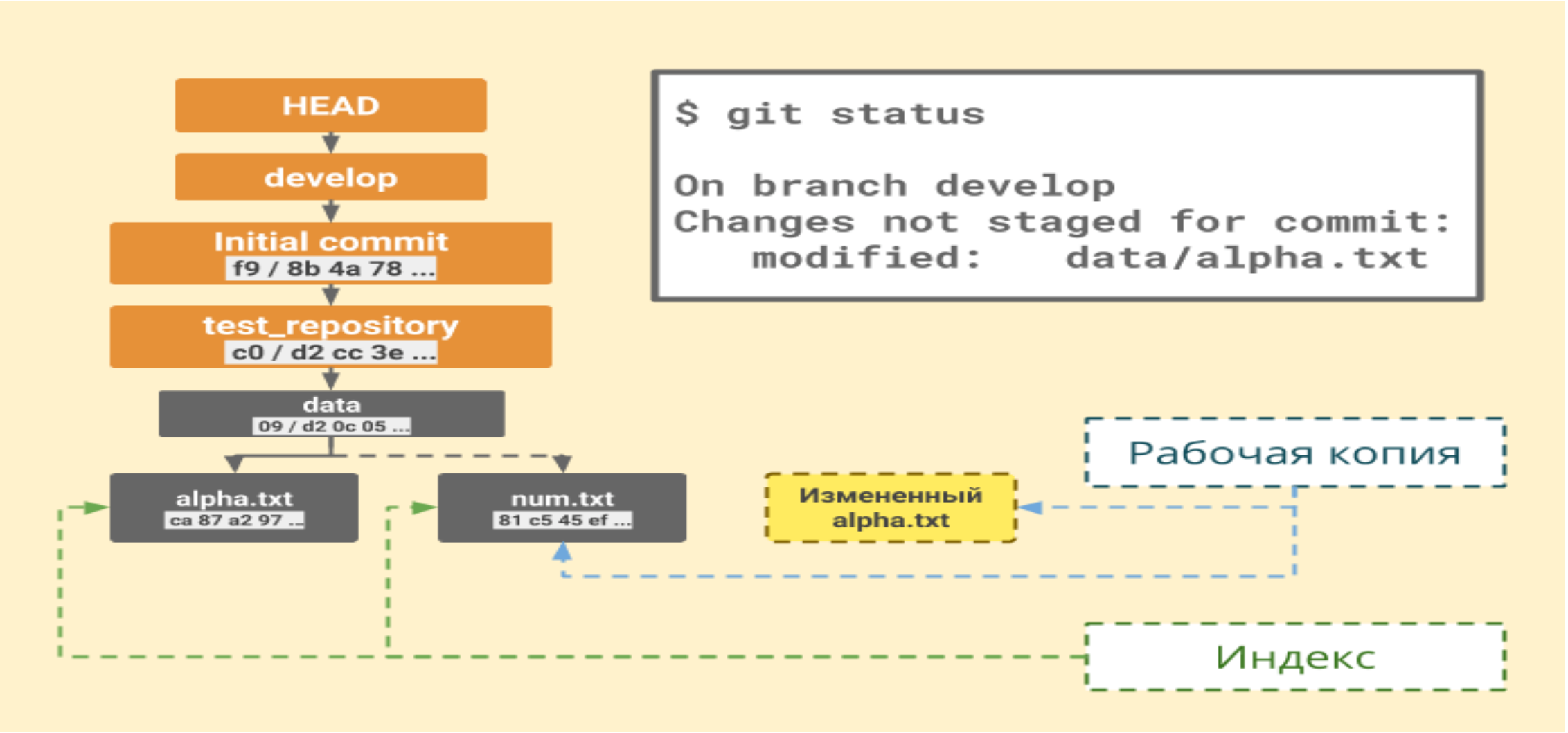
GIT commit – Этап 3. Направить ветку на текущий коммит



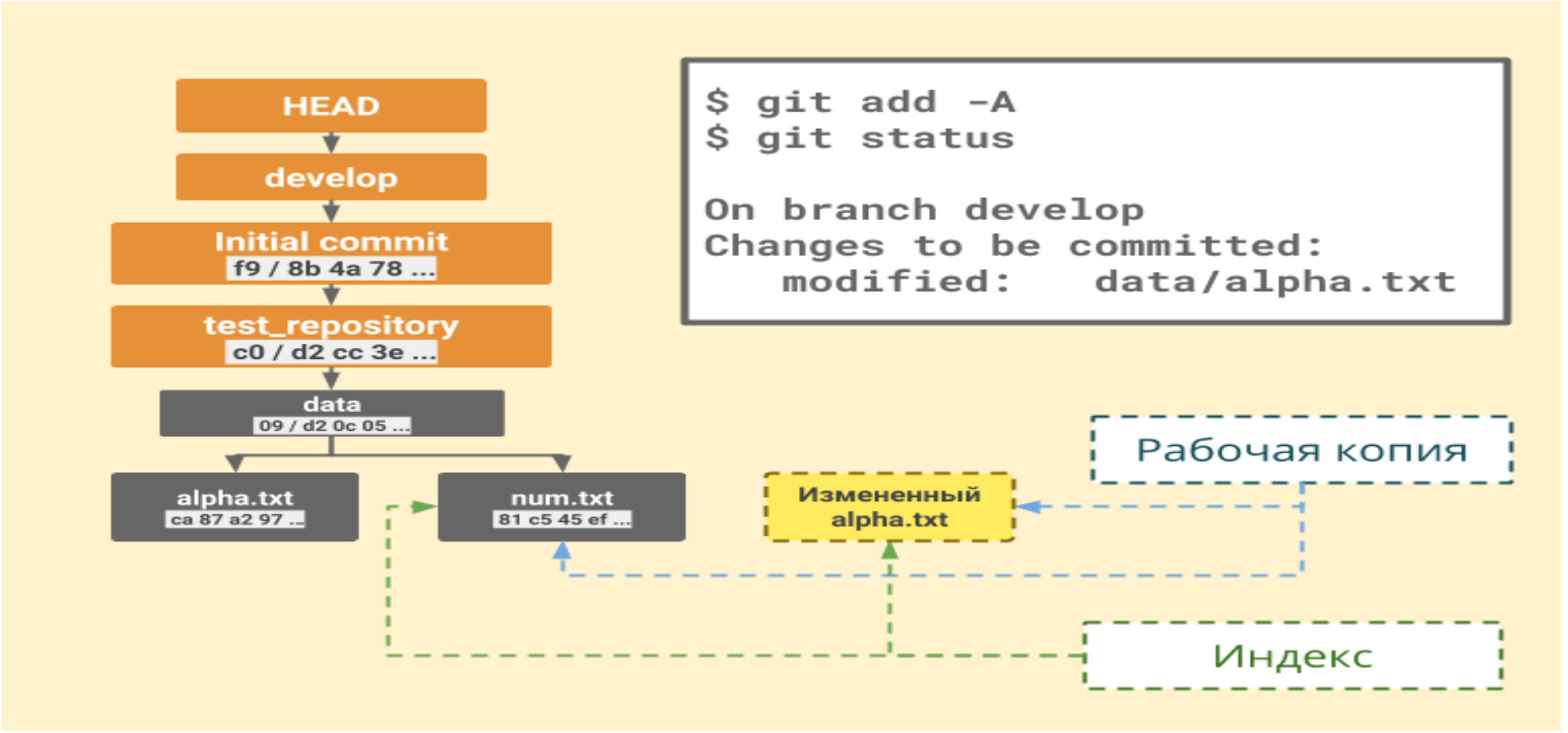
GIT commit – граф второго коммита (сейчас)



GIT commit – граф второго коммита (изменения без add)

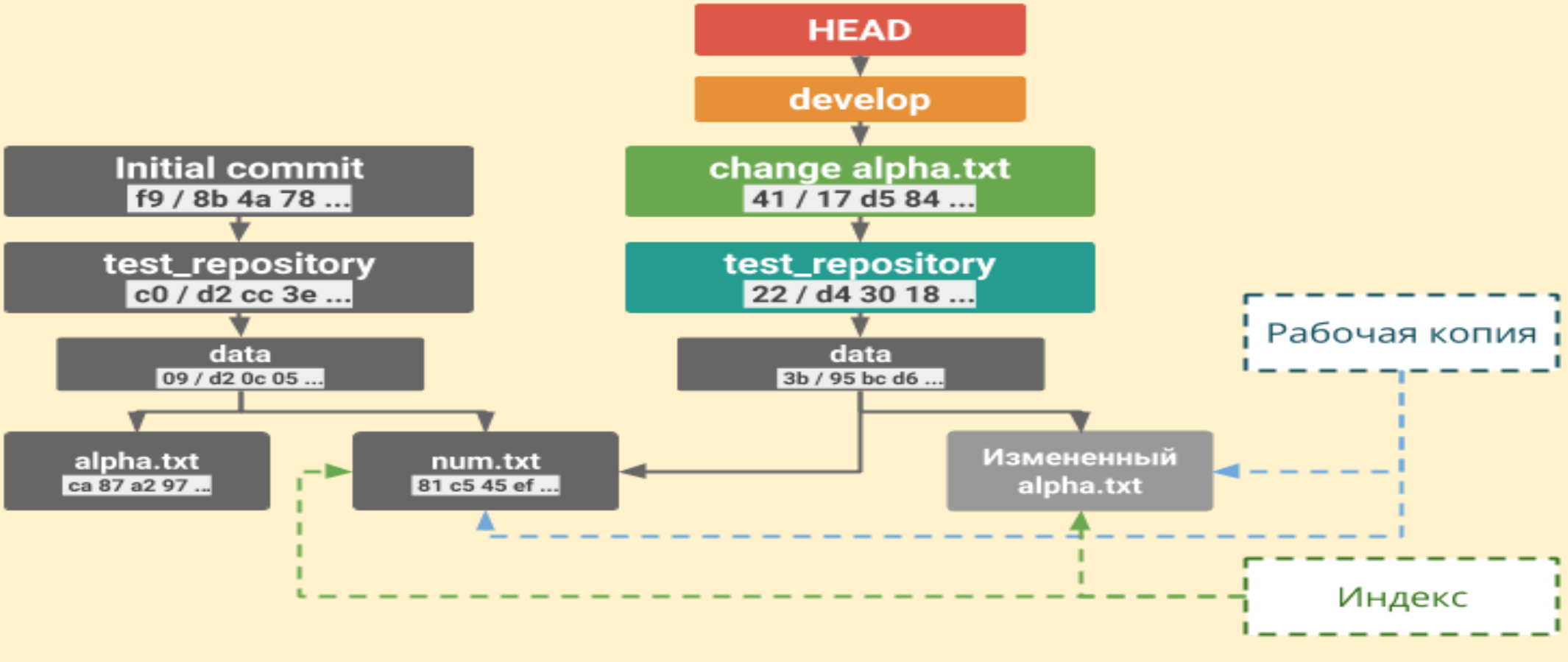


GIT commit – граф второго коммита (изменения после add)



GIT commit – граф второго коммита (итого)

```
$ git commit -m "change alpha.txt"
[develop 4117d58] change alpha.txt 1 file changed
```



Git add как можно делать

Команда	Результат
git add <список файлов> git add file1 file2	Добавить конкретные файлы
git add .	Добавить все файлы в текущей папке
git add *.java	Добавить все файлы в текущей папке с расширением .java
git add someDir/*.java	Добавить все файлы в папке someDir с расширением .java
git add someDir/	Добавить все файлы в папке someDir
git add “*.java”	Добавить все файлы в проекте с расширением .java

Git diff – история изменений

Команда	Результат
git diff	Показывает разницу между текущим неотслеживаемым состоянием репозитория и последним снимком репозитория
git diff --staged	показывает разницу между текущим отслеживаемым состоянием репозитория и последним снимком репозитория
git diff COMMIT_ID	показывает разницу между текущим состоянием репозитория и указанным снимком репозитория

Git rm – удаление файлов

Команда	Результат
git rm <ключ> <имя файла>*	удаляет файл из рабочей копии и staging area / только из staging area . Данная команда не может удалить файл только из рабочей копии.
git rm -f | --force <имя файла>	удаление без предупреждения
git rm --cached <имя файла>	удаление файла только из staging area, в рабочем каталоге файл останется

* - можно указывать маски файлов

Git commit – изменение последнего коммита

Команда	Результат
git commit --amend -m "Updated message for the previous commit"	**Изменить сообщение в коммите**
git add dir2 **git commit --amend --no-edit**	**Добавить что-то в текущий коммит**

Внимание! Не изменяйте публичные коммиты.

С помощью amend прекрасно исправляются локальные коммиты, а исправления можно передать в общий репозиторий. Однако изменять коммиты, уже доступные другим пользователям, не следует. Помните, что изменённые коммиты являются совершенно новыми, а предыдущий коммит уже не будет доступен в текущей ветке. Последствия будут такими же, как при отмене изменений публичного снимка.

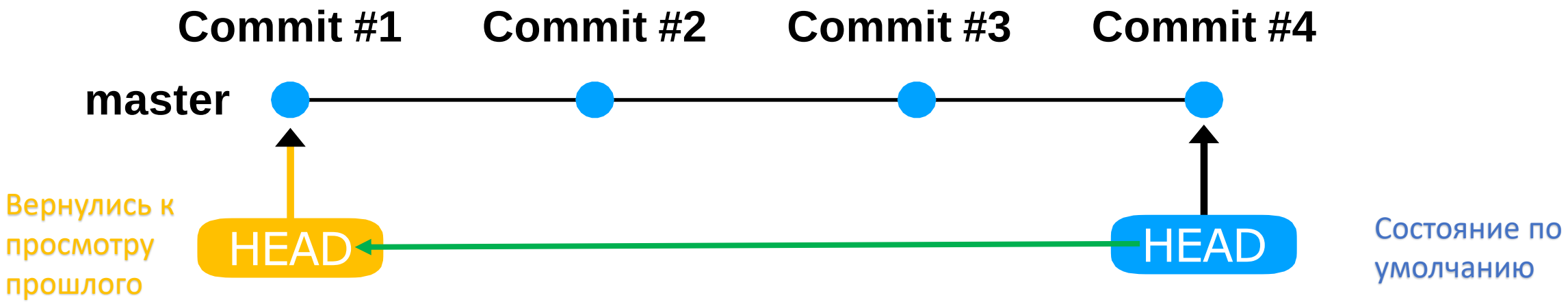
Git откат коммитов

Команда	Результат
git revert HEAD	Откатиться к предыдущему коммиту
git revert <COMMIT_ID>	Откатиться на конкретный коммит

git revert – создает новый коммит без изменения истории
git rebase – удаляет историю

Git checkout – описание (переход между коммитами)

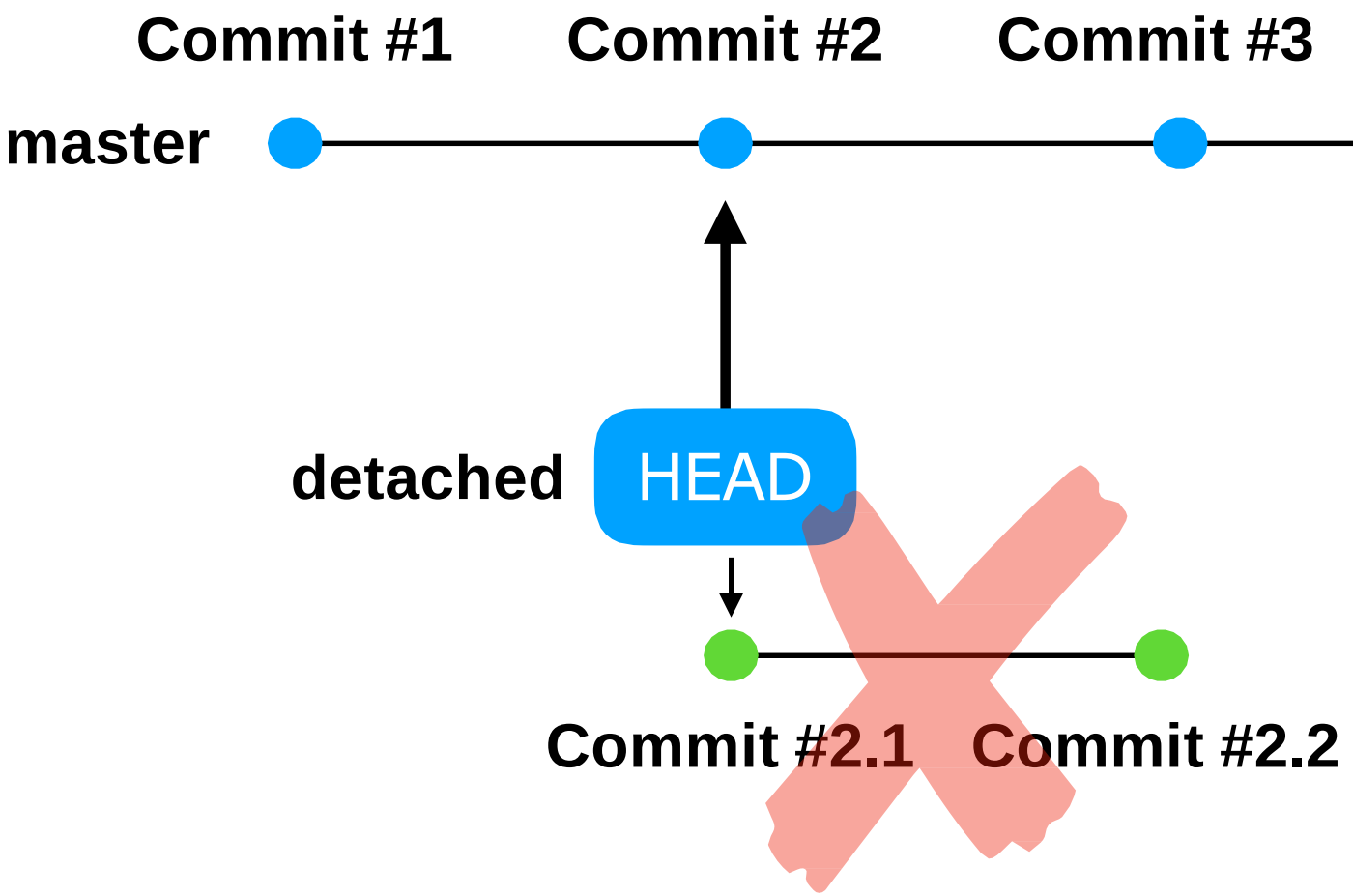
Используется для перемещения между коммитами, версиями отдельных файлов и ветками



- **Состояние проекта полностью вернулось к указанному снимку. При этом никакие коммиты не удалились. Мы в любой момент можем перенестись обратно в актуальную версию.**

- **Указатель HEAD находится в состоянии DETACHED (рус. отделенный). Он отделен от актуальной версии проекта. Любые изменения или коммиты сделанные в этом состоянии удаляются сборщиком мусора при переходе к другому коммиту.**

Git checkout – DETACHED состояние



!Нужно использовать технологию бранчевания для таких задач

Git checkout – варианты использования (переход между коммитами)

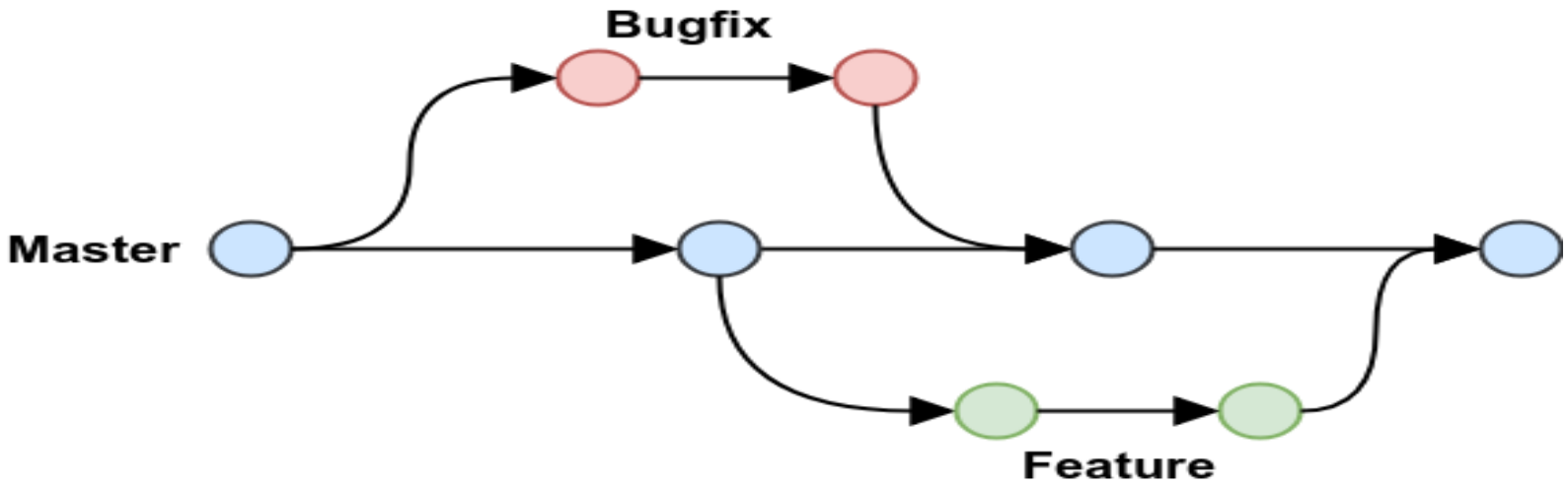
Команда	Результат
git checkout <хэш commit>	Переход к конкретному коммиту в ветке
git checkout HEAD^^ или git checkout HEAD~2	Переход на два коммита назад
git checkout <имя ветки>	Вернуться к последнему коммиту в ветке

Git checkout – варианты использования (перемещение между файлами)

Команда	Результат
git checkout <указатель коммита> -- путь_до_файла_1 путь_до_файла_2	Возвращает два файла к версии, которая была у них в указанном коммите
git checkout -- путь_до_файла_1 путь_до_файла_2	Возвращает два файла к последней версии коммита (HEAD). Работает только для untracked/modified* файлов
git checkout -- .	Возвращает все файлы к последней версии коммита (HEAD). Работает только для untracked/modified* файлов

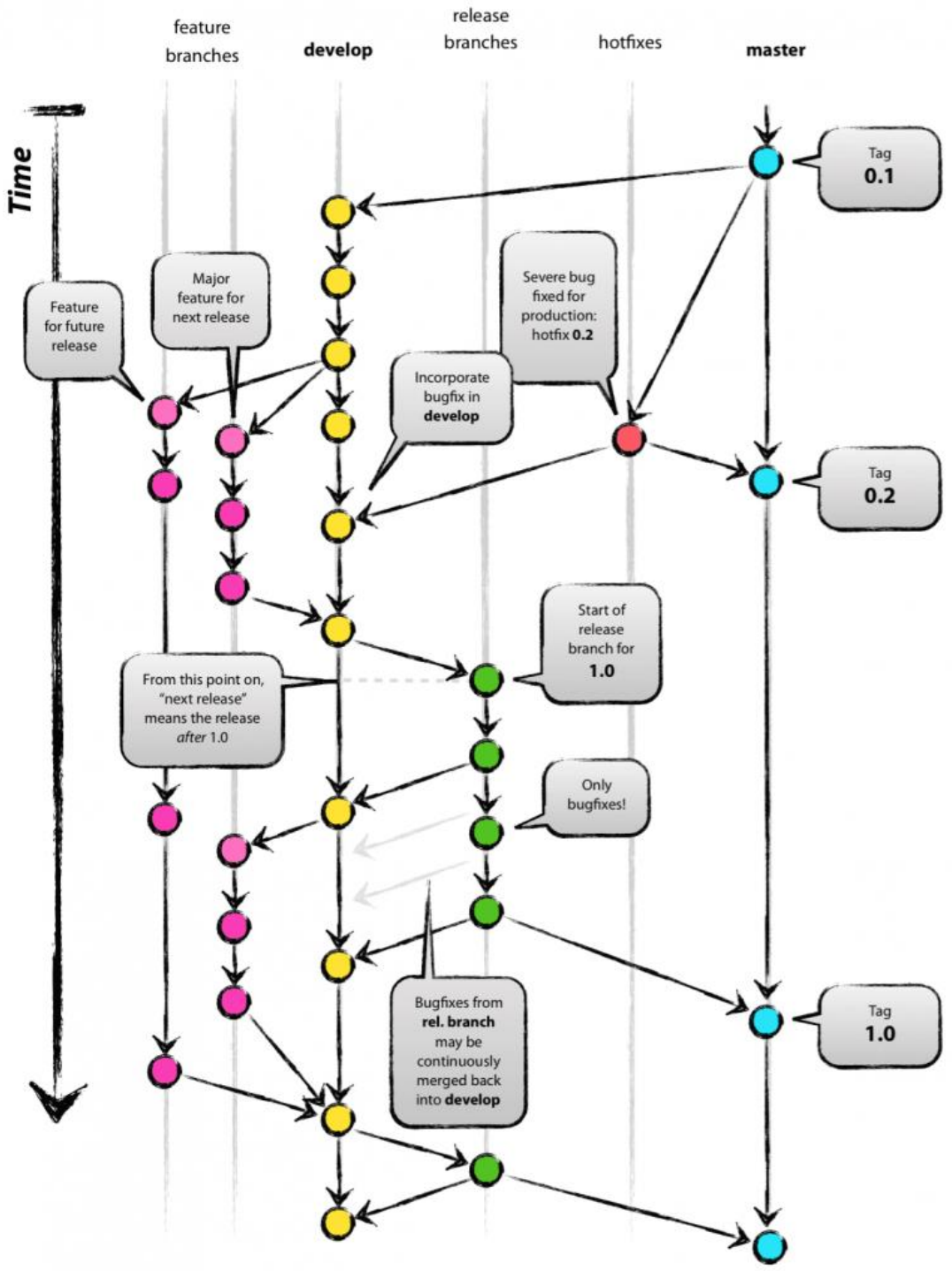
* - если мы хотим «откатить» отслеживаемые файлы, необходимо их сбросить до неотслеживаемых через git reset

Git – работа с ветками

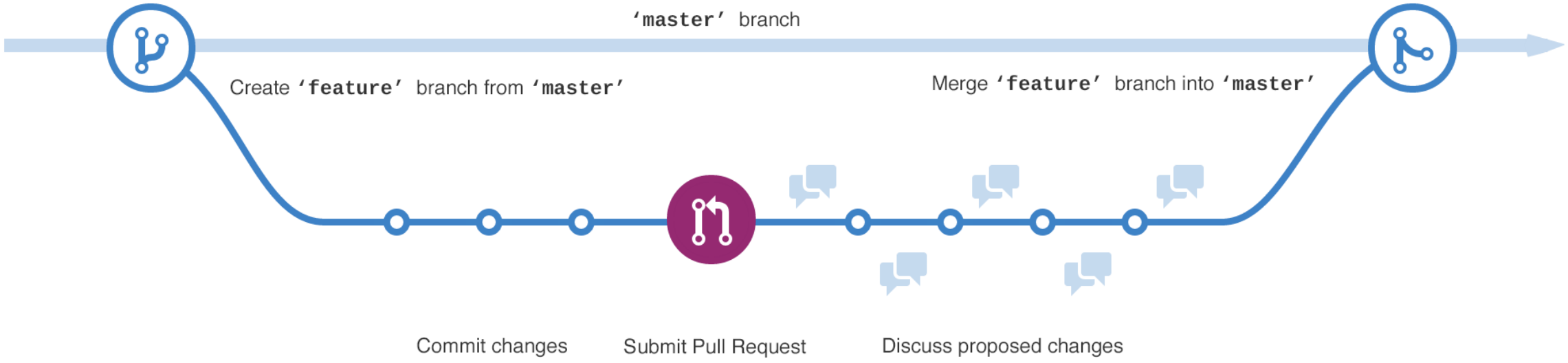


- Новые функции разрабатываются в отдельных ветках
- Ветка master/release/production содержит стабильную версию проекта
- Сразу несколько разработчиков могут работать в своих ветках над своими задачами. После завершения работы над задачами, эти ветки "сливаются" в “master” ветку.

Git Flow


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

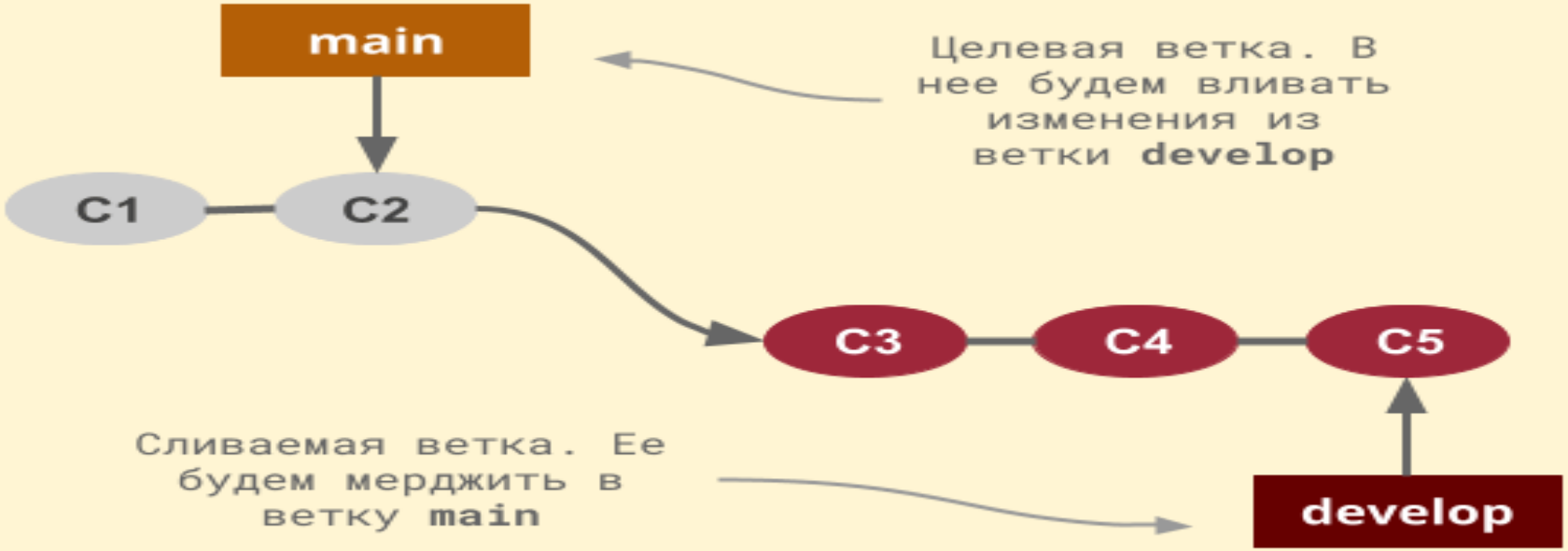
GitHub Flow


'master' branch
Create 'feature' branch from 'master'
Merge 'feature' branch into 'master'
Commit changes
Submit Pull Request
Discuss proposed changes

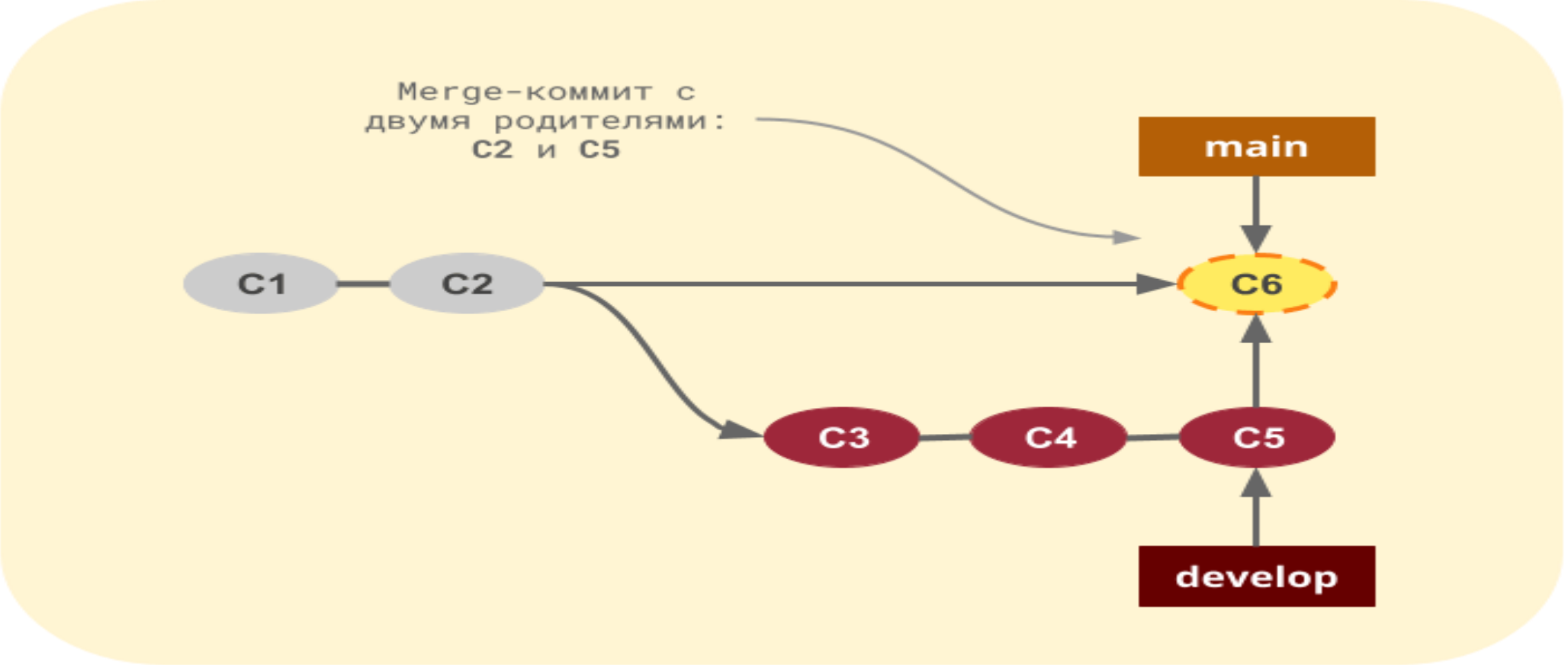
Работа с ветками, базовые команды

Команда	Результат
git branch	Просмотр на какой ветке мы находимся
git branch <name-branch>	Создание новой ветки с именем name-branch
git branch –d <name-branch>	Удаление ветки
git checkout <имя ветки>	Вернуться к последнему коммиту в ветке
git merge <FROM branch-name>	Добавляем коммиты ветки branch FROM в текущую ветку Правильно делать: Git checkout <TO branch-name> Git merge <FROM branch-name>

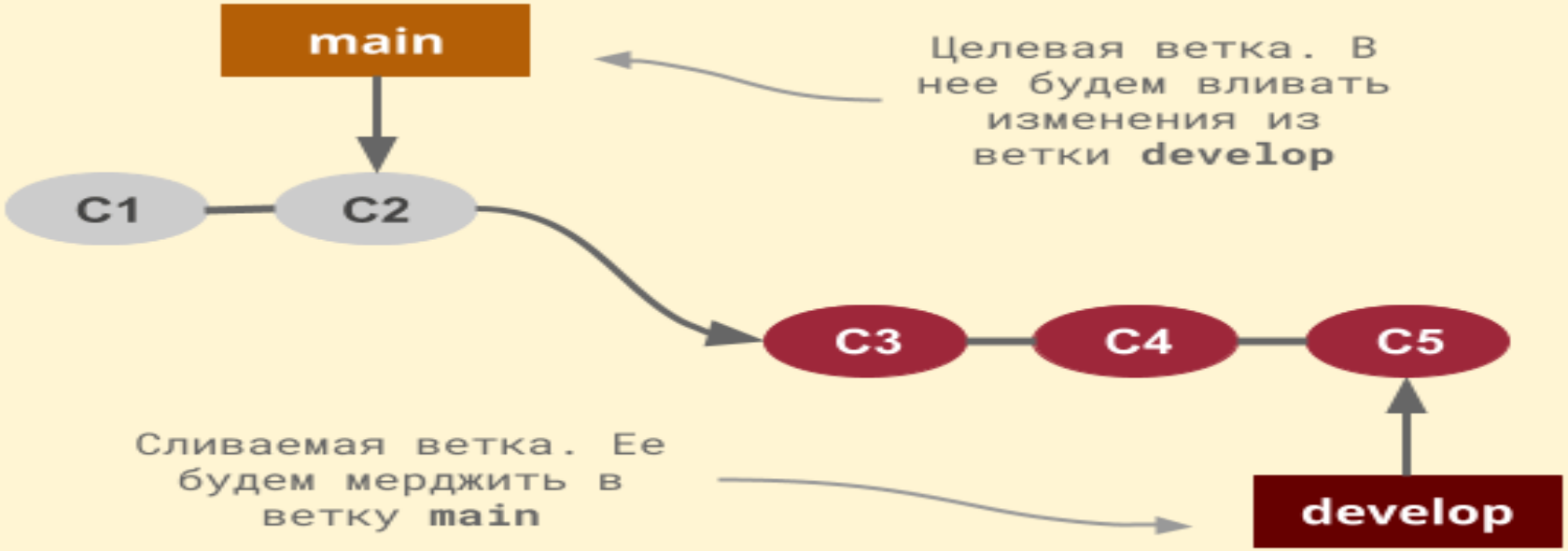
Git merge - явное слияние (ДО)



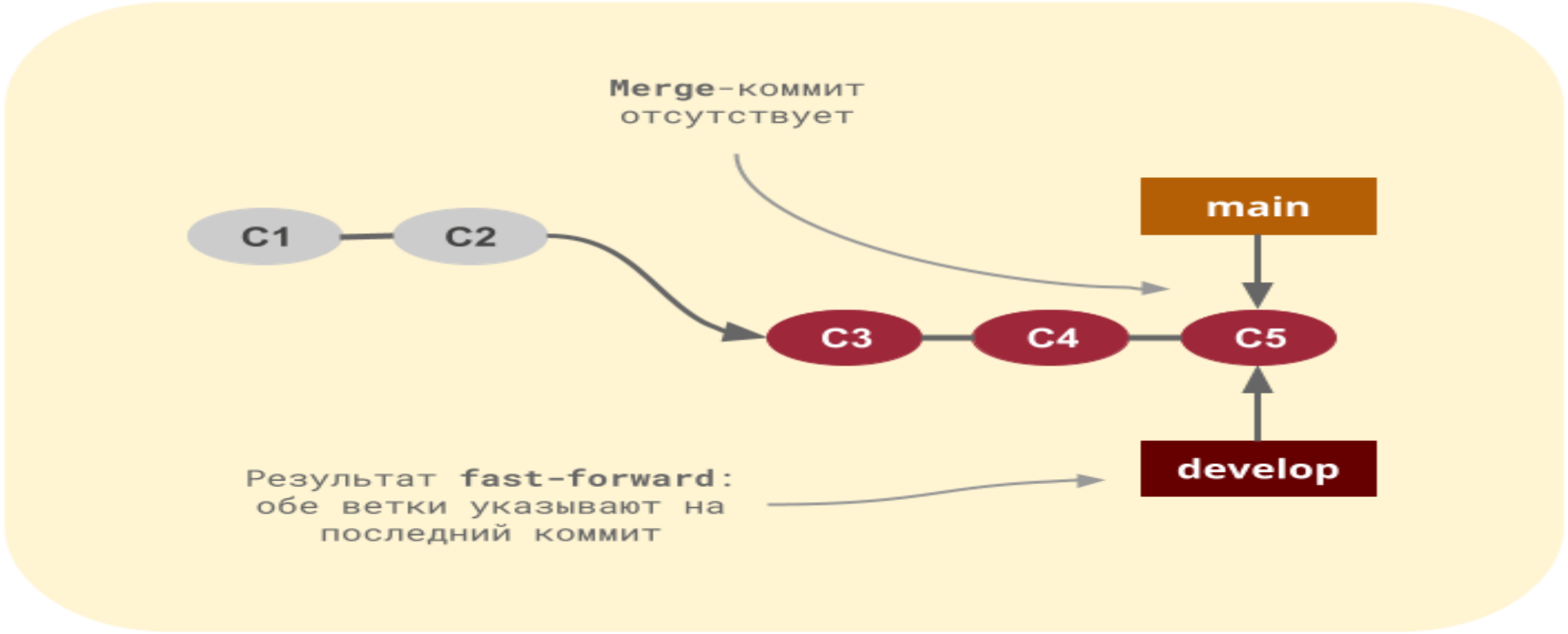
Git merge - явное слияние (РЕЗУЛЬТАТ)



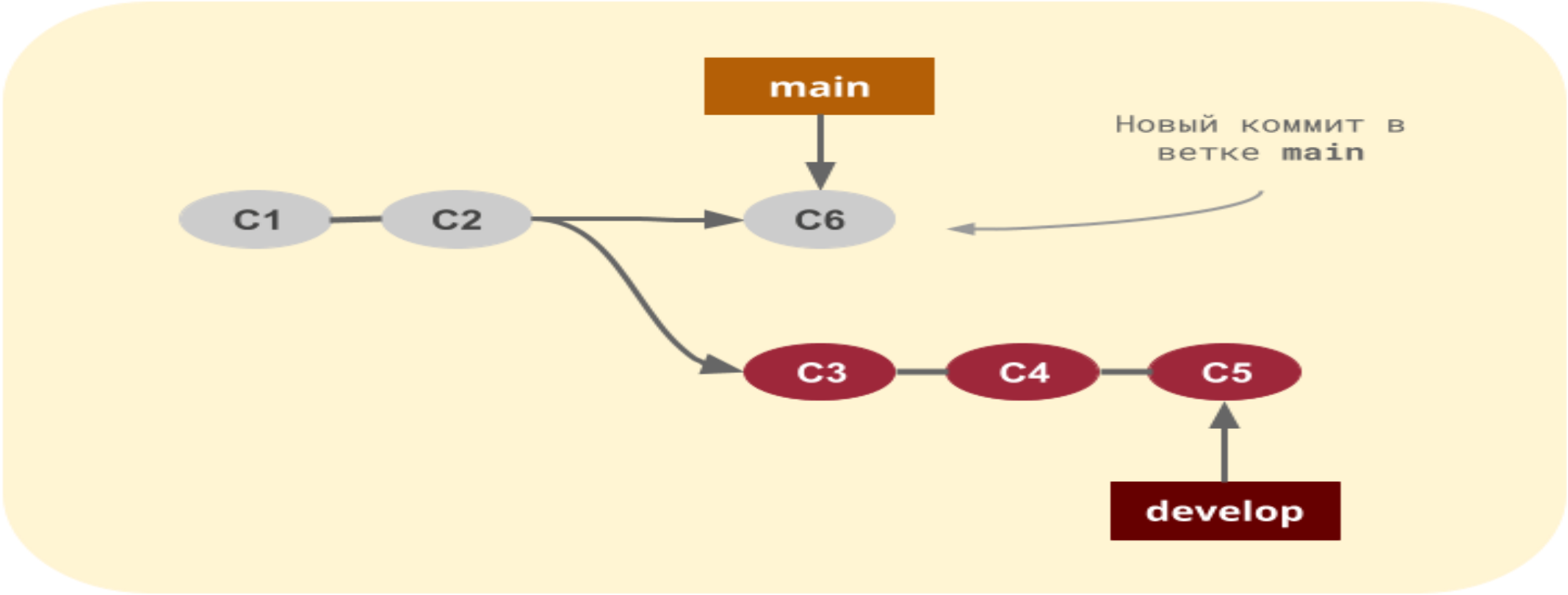
Git merge – неявное слияние (ДО)



Git merge – неявное слияние (РЕЗУЛЬТАТ)

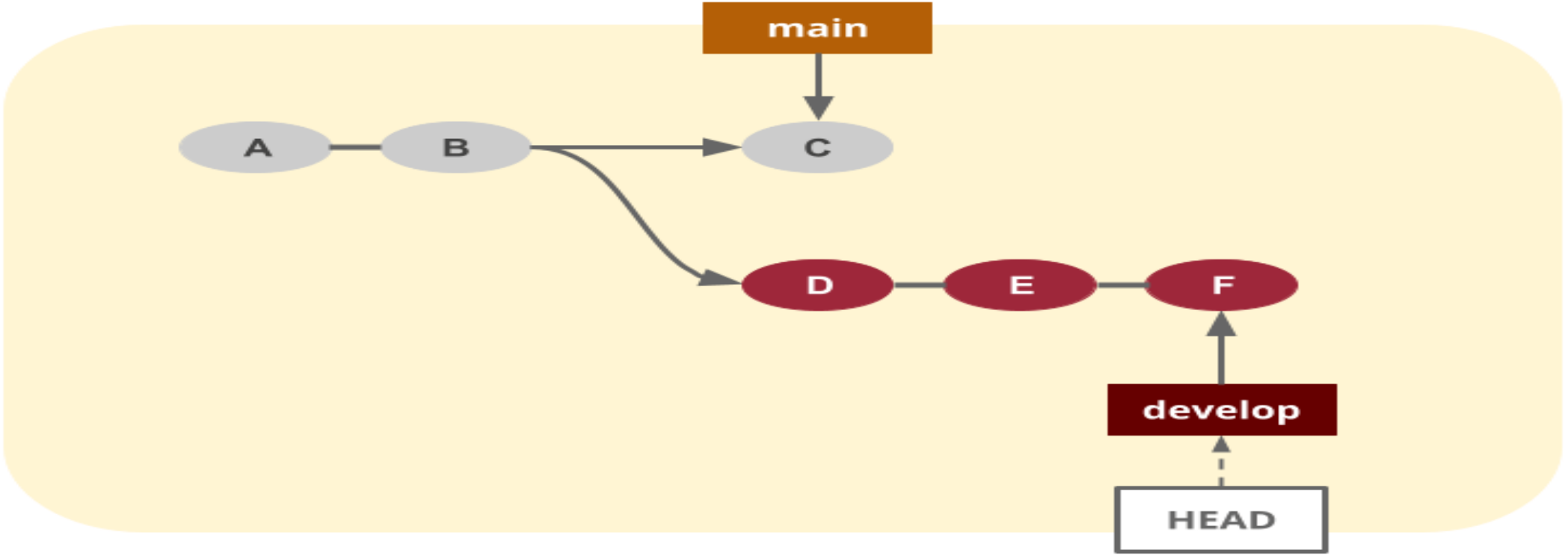


Git merge – итоги

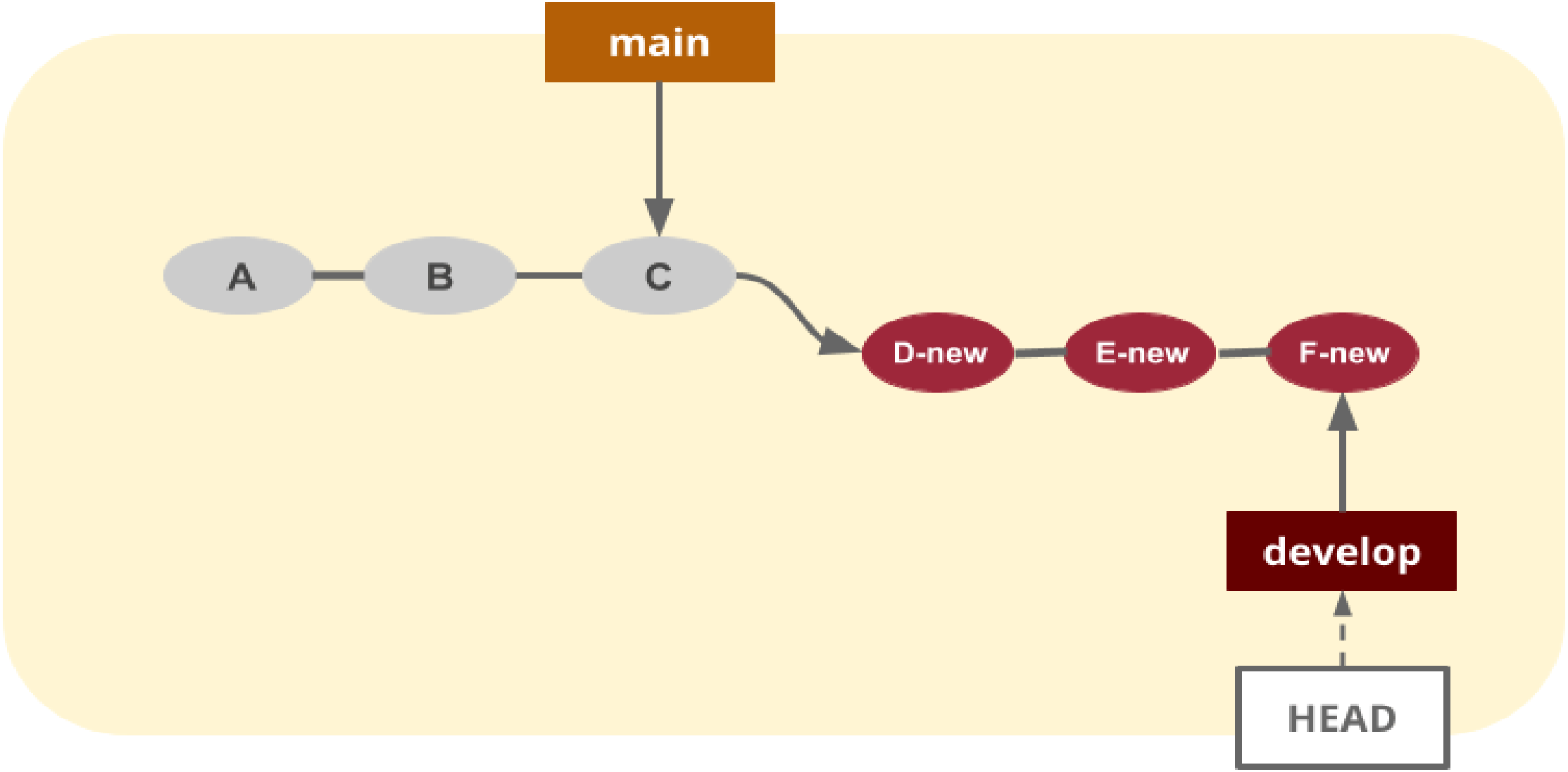


Git rebase – альтернатива merge

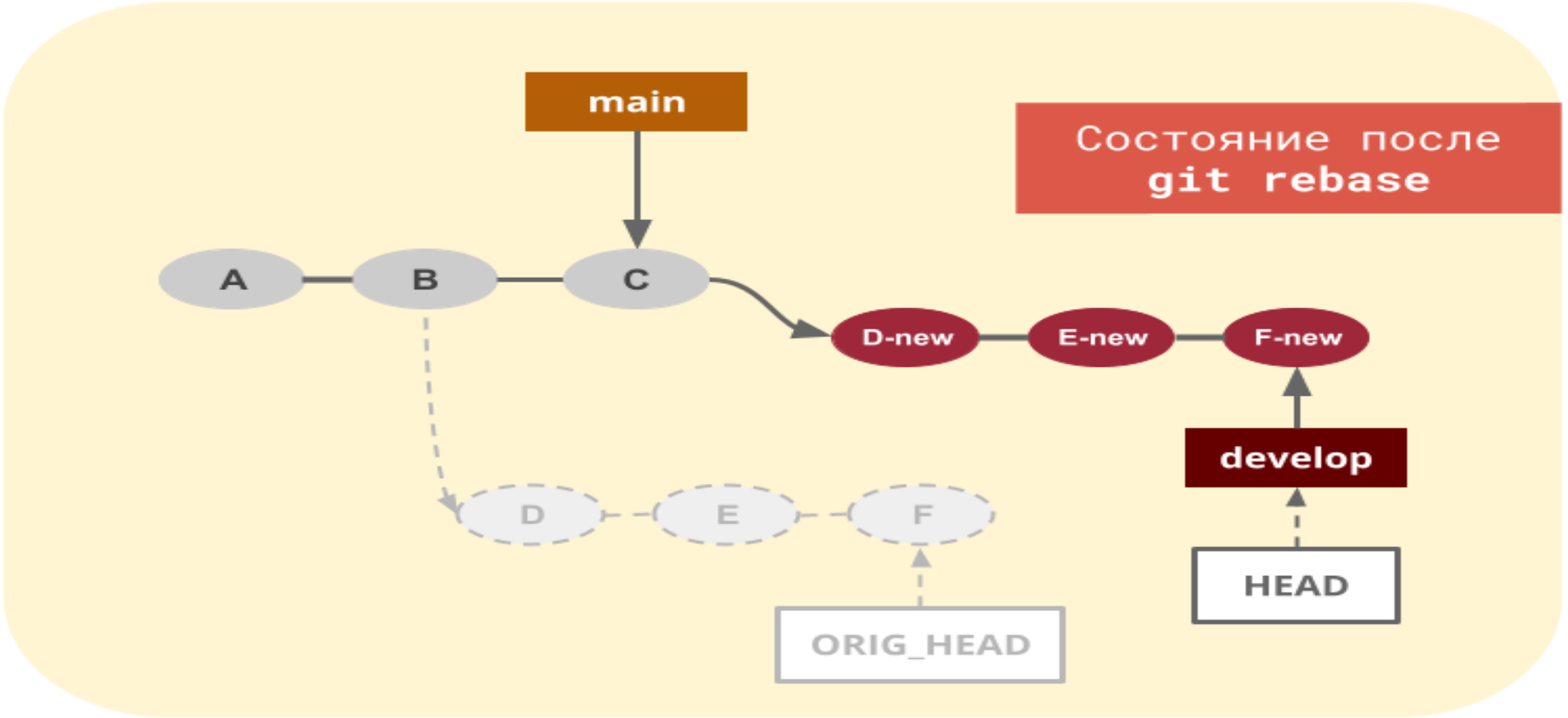
Команда	Результат
git rebase <целевая ветка>	Перемещает все коммиты: от общего коммита двух веток до последнего коммита текущей ветки на вершину переданной ветки.



Git rebase – результат git rebase main



Git rebase – что произошло



Git rebase – интерактивный режим

Команда	Результат
git rebase -i HEAD~3	**Перемещает все коммиты: от общего коммита двух веток до последнего коммита текущей ветки на вершину переданной ветки.**

p, **pick <коммит>** – просто использовать коммит, ничего не менять
r, **reword <коммит>** – использовать коммит, но поменять его сообщение
e, **edit <коммит>** – использовать коммит, но остановить ребейз, чтобы добавить в коммит больше файлов
s, **squash <коммит>** – использовать коммит, объединив его с предыдущим
f, **fixup <коммит>** – как **squash**, но удаляет информацию об объединенном коммите из истории.

Git cherry-pick

Команда	Результат
git cherry-pick <хеш коммита>	Берет переданный коммит и создает в текущей ветке его точную копию.
git cherry-pick <хеш первого коммита> … <хеш последнего коммита>	Также в команду можно передать первый и последний коммит последовательности, тогда та же операция будет выполнена для всех коммитов последовательности.

Git merge conflict


```
If you have questions, please
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> branch-a
```

Здесь состояние ветки TO

Здесь состояние ветки FROM

Дополнительные возможности

Последние комиты во всех ветках

- git branch –v

Графическое представление коммитов

- git log --graph --oneline --decorate

Не отправлять какие-то файлы в git

- git ignore
- git rm –cached

Обнуление истории

- git reset
- git revent

Тэги

- git tag -a v.1.0.1 -m "Version«
- git push [branch] [tag]

git/hooks

git config --global commit.template /.template.txt

Stash и compare

Смотрим разницу между ветками

- git log master..tests
- git log ^master tests
- git log test not master
- git diff –name-status branch1 branch2 Комиты, которые есть в одной ветке, но нет в обоих

Сохраняем «dirty file» в стэке

- git status
- git stash
- git stash list
- git stash apply
- git stash drop stash@{1}
- git stash pop -> apply and drop

Git hooks

Клиентские — находятся на машине конечного. Каждый контрибьютор настраивает хуки только для себя в своей копии существующего репозитория, и они не повлияют на других контрибьюторов. Для того чтобы добавить хук, достаточно изменить файл в директории $PROJECT_DIR/.git/hooks/. Изменение в этой директории не может быть закоммичено и будет влиять только на текущего пользователя. Такие хуки можно игнорировать, используя дополнительный флаг git commit --no-verify.

Серверные —они будут применяться ко всем контрибьюторам проекта.
Для установки такого хука также достаточно изменить файл в директории $PROJECT_DIR/hooks/, но делать это нужно в удаленном репозитории. Пропустить исполнение таких хуков нельзя. Также при клонировании репозитория не получится выкачать хуки, они останутся на удаленном репозитории.

Git hooks – простой пример

commit-msg.sample -> commit-msg

```
commitRegex='^(MFTI-[0-9]+|merge|hotfix)’
if ! grep -qE "$commitRegex" "$1";  then
        echo "Aborting according commit message policy.
Please specify JIRA issue MFTI-XXXX."          exit 1
fi
```

Git hooks – еще примеры

Запрет на push в ветку

pre-receive

```
#!/bin/bash

changedBranch=$(git symbolic-ref HEAD | sed -e 's,.*/\(.*\),\1,')
blockedUsers=(junior1 junior2)

if [[ ${blockedUsers[*]} =~ $USER ]]; then
    if [ $changedBranch == "master" ]; then
        echo "You are not allowed commit changes in this branch"
        exit 1
    fi
fi
```

Git hooks – еще примеры

printf shall not pass

pre-receive, pre-commit

```
#!/bin/bash

blackList="console.info\|console.log\|alert\|var_dump"
result=0
while read FILE; do
   # check that file not removed(also can be implemented using --diff-filter)
   if [[ -f $FILE ]]; then
      if [[ "$FILE" =~ ^.+(php|html|js)$ ]]; then
         RESULT=$(grep -i -m 1 "$blackList" "$FILE")
         if [[ ! -z $RESULT ]]; then
            echo "$FILE contains denied word: $RESULT"
            result=1
         fi
      fi
fi
done << EOT
   $(git diff --cached --name-only)
EOT

if [ $result -ne 0 ]; then
   echo "Aborting commit due to denied words"
   exit $result
fi
```

Работаем с удаленным репозиторием

Клонируем

- git clone URL

Смотрим удаленные ветки

- git remote -v

Делаем изменения и отправляем их на сервер

- git push [remote-name] [branch]
- git push origin nameLocal:nameRemote

Получаем обновления с сервера

- git fetch –all
- git pull