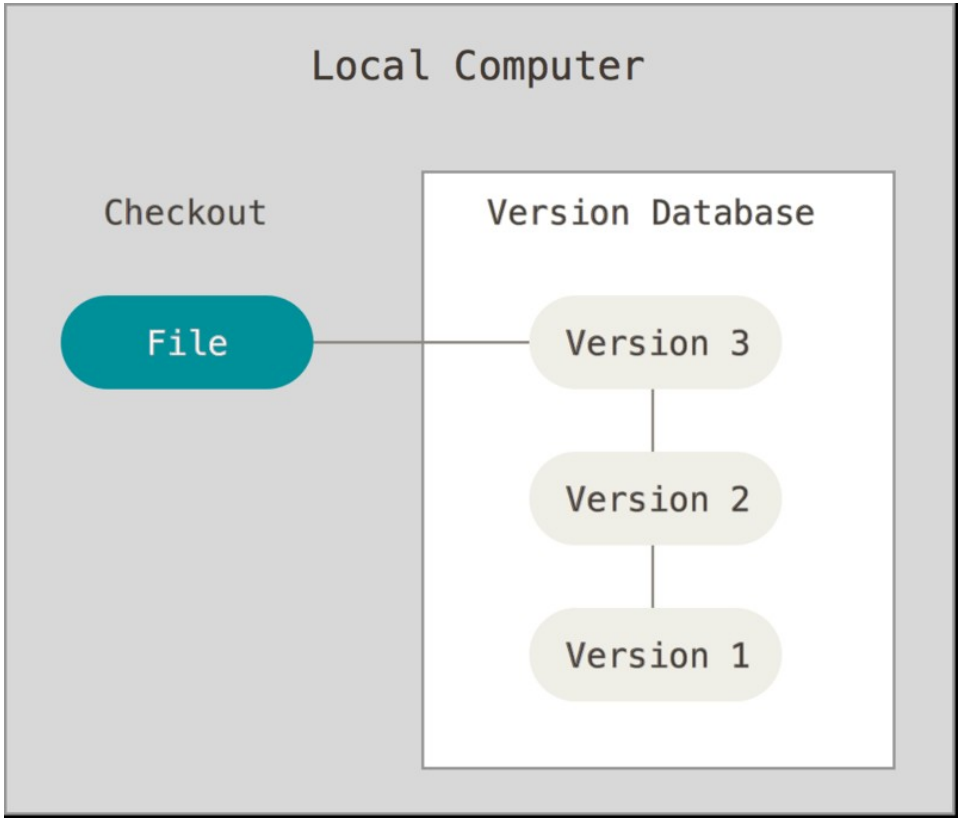
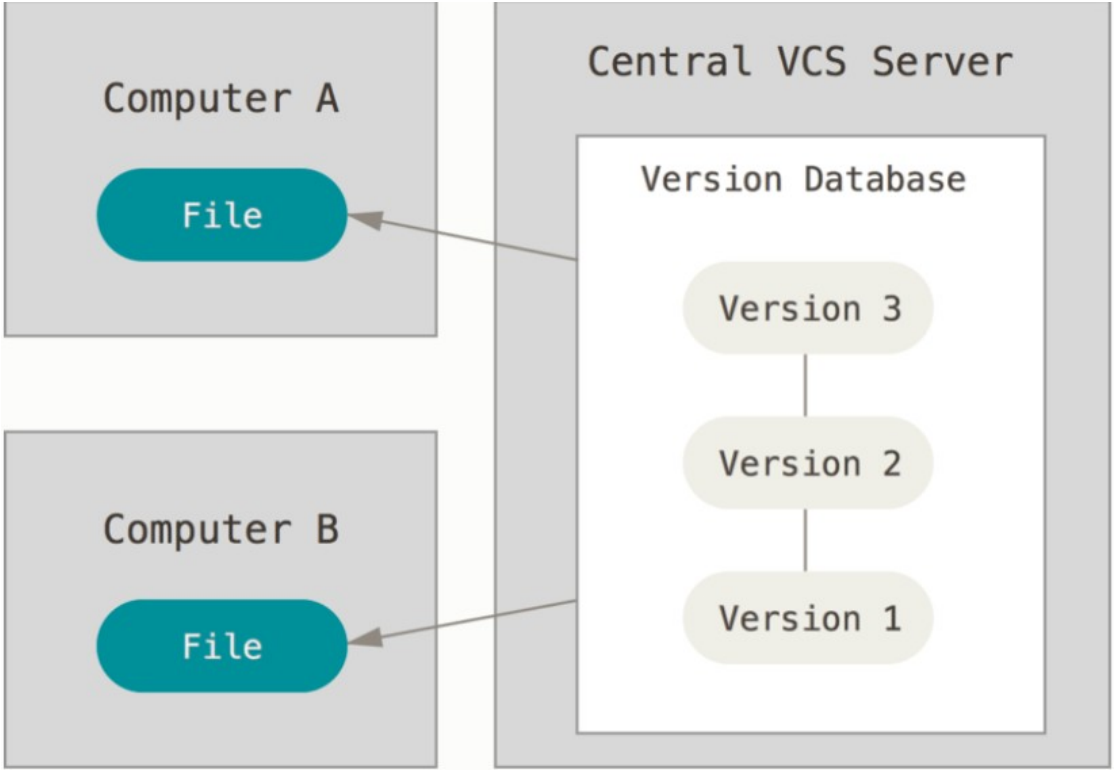
Система контроля версий Git

Локальные VCS



Централизованные VCS

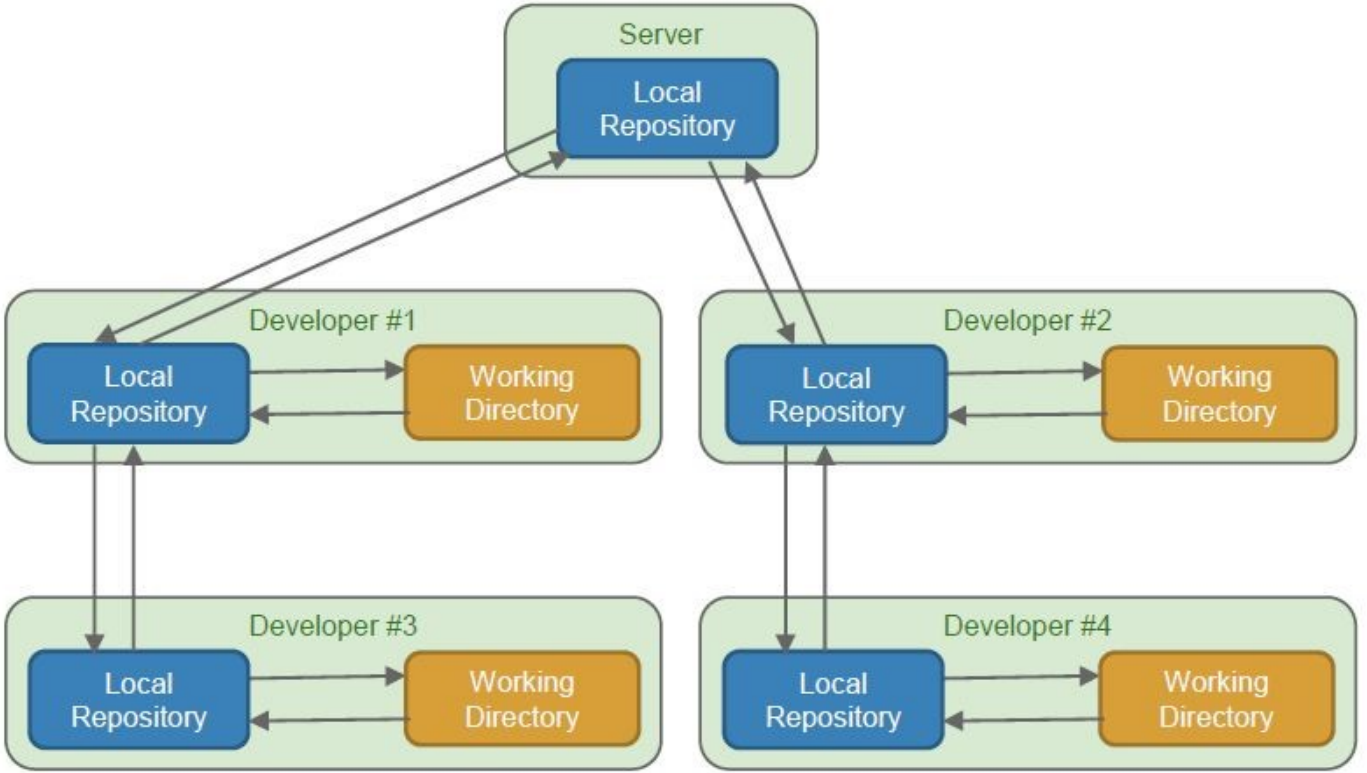


Например: Subversion и Perforce

Централизованные VCS

- Достоинства
  - Централизованное администрирование
  - Привычный workflow
  - Управление правами доступа
- Недостатки
  - Единая точка отказа – сервер
  - Любые изменения влияют на всех пользователей
  - Неудобная работа с ветками – легко создать, тяжело смержить (merge)
  - Возможны блокировки - захват пессимистической блокировки одним пользователем

Распределенные VCS



Например: Git, Mercurial, Bazaar

Распределенные VCS

- Достоинства
  - Гибкая работа с ветками
  - Автономность (как каждого разработчика, так и от сервера вообще)
  - Сборка артефактов отделена от разработки
  - Локальные операции работают быстро
  - Разделены операции фиксации изменений (commit) и публикации изменений (push)
- Недостатки
  - В каждой копии необходимо хранить всю историю изменений (иногда считается плюсом)
  - Требуется более тщательное управление доступом (иногда считается плюсом)
  - Сложны в использовании

Знакомство с Git

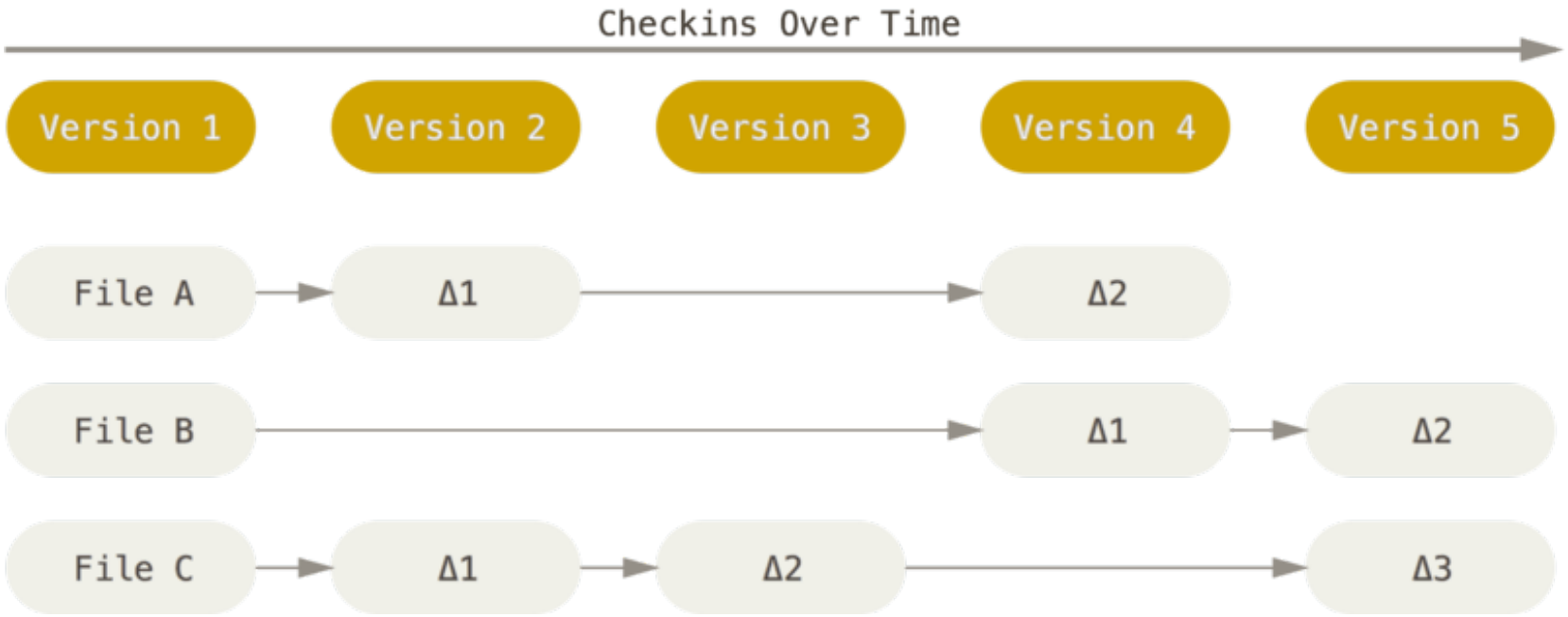
Git — распределенная система контроля версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux. Первая версия была выпущена 7 апреля 2005 года. Сейчас его поддерживает Джунио Хамано.

Целостность данных

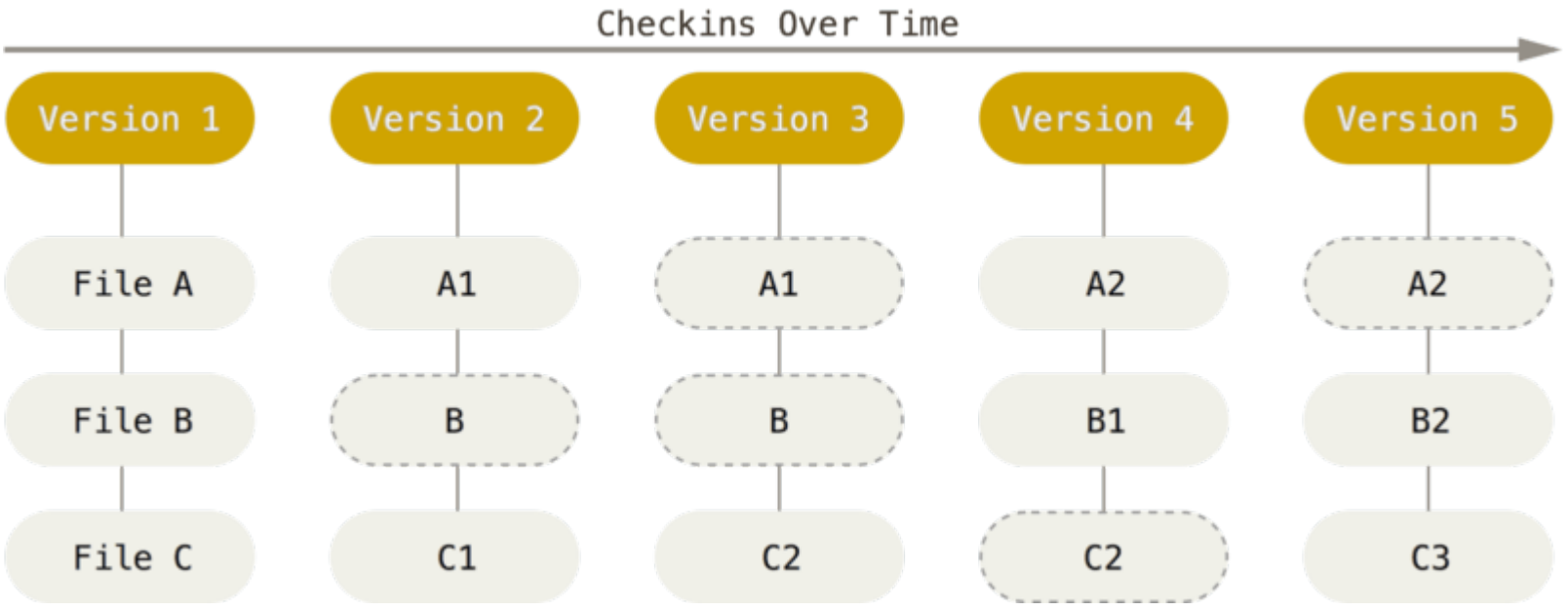
Перед сохранением любого файла Git вычисляет контрольную сумму, и она становится индексом этого файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Эта функциональность встроена в сам фундамент Git'а и является важной составляющей его философии. Если информация потеряется при передаче или повредится на диске, Git всегда это выявит.

Механизм, используемый Git'ом для вычисления контрольных сумм, называется SHA-1 хешем. Это строка из 40 шестнадцатеричных символов (0-9 и a-f), вычисляемая в Git'е на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит примерно так: 24b9da6552252987aa493b52f8696cd6d3b00373

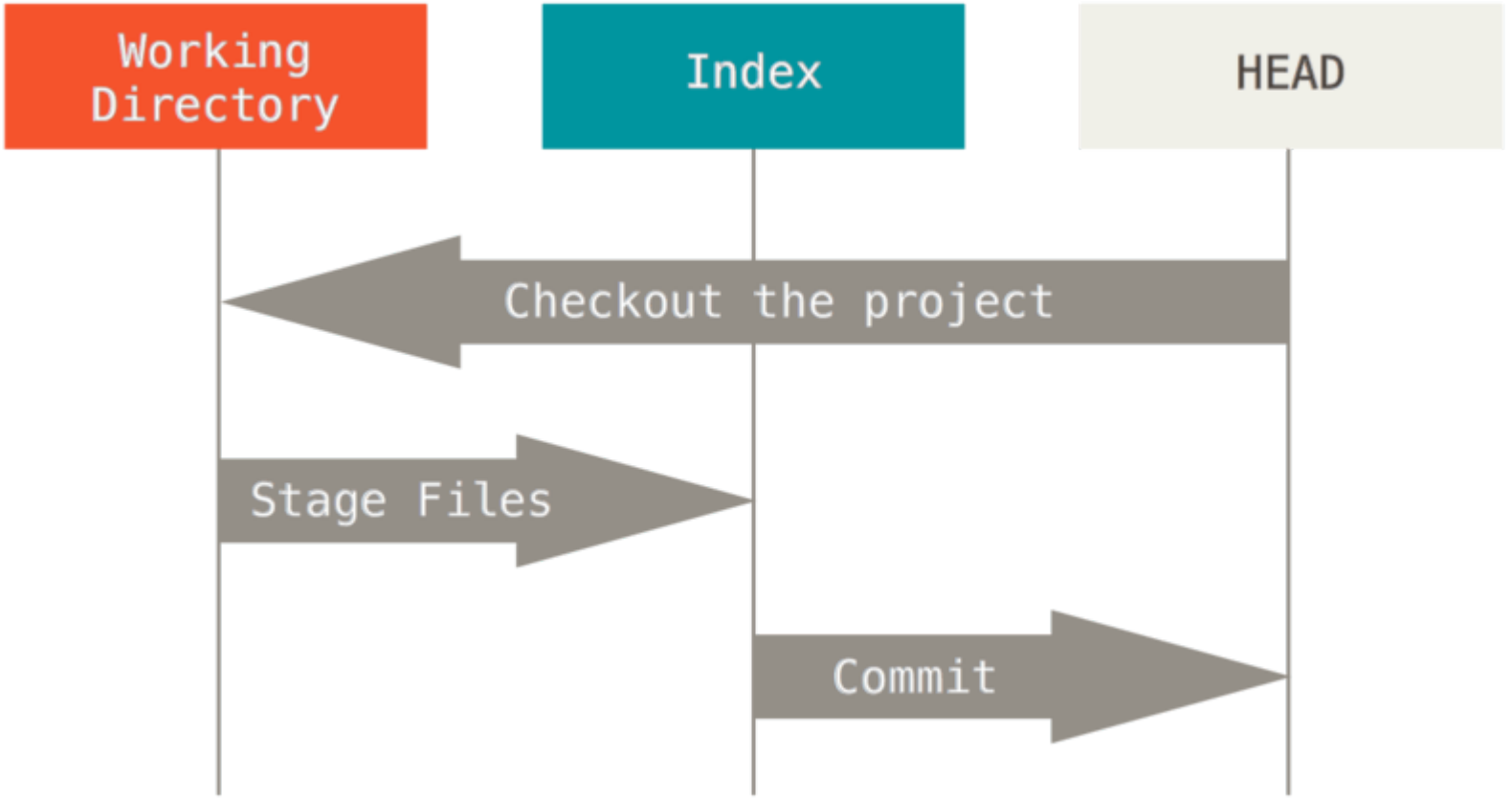
Различия


Checkins Over Time
Version 1
Version 2
Version 3
Version 4
Version 5
File A
Δ1
Δ2
File B
Δ1
Δ2
File C
Δ1
Δ2
Δ3

Поток снимков



Три состояния



Установка Git

- Linux: sudo apt-get install git
- OS X: brew install git
- Windows: https://git-scm.com/downloads

Настройка Git

- Файл ~/.gitconfig хранит настройки конкретного пользователя. Этот файл использует при указании параметра --global
- Первое, что вам следует сделать после установки Git'a, — указать ваше имя и адрес электронной почты:
  - $ git config --global user.name "John Doe"
  - $ git config --global user.email johndoe@email.com

Настройка Git

- По умолчанию Git использует стандартный редактор вашей системы, обычно это Vi или Vim. Если вы хотите использовать другой текстовый редактор можно сделать следующее:
  - $ git config --global core.editor nano
- Проверить используемые настройки, использовать команду:
  - $ git config --list --global
- Как получить помощь:
  - $ git help <comand>

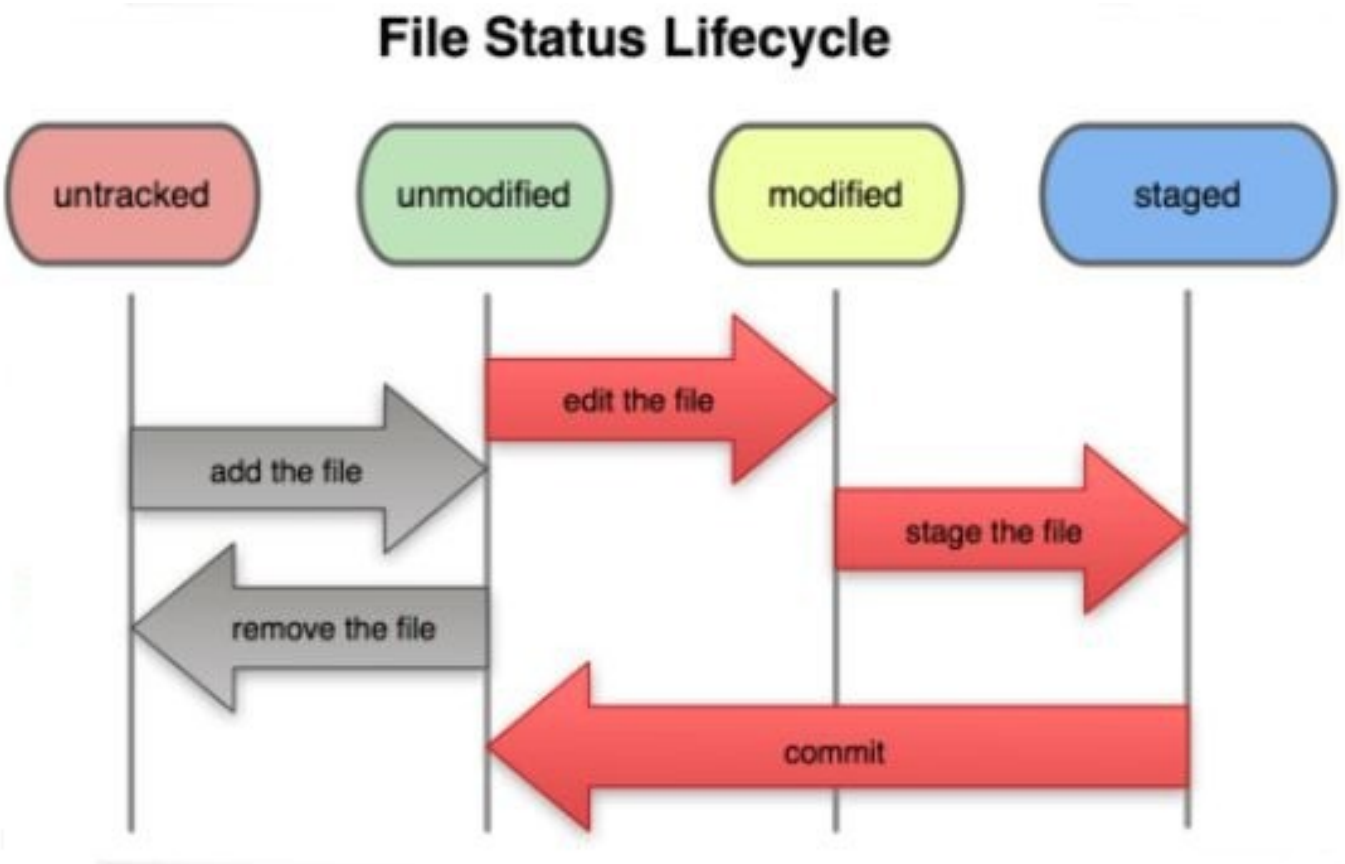
Файл **.gitignor**

- Зачастую, у вас имеется группа файлов, которые вы не только не хотите автоматически добавлять в репозиторий, но и видеть в списках неотслеживаемых. К таким файлам обычно относятся автоматически генерируемые файлы (различные логи, результаты сборки программ и т.п.). В таком случае, вы можете создать файл **.gitignore** с перечислением шаблонов соответствующих таким файлам. Вот пример файла **.gitignore**:
  - $ cat .gitignore
  - *.[oa]
  - *~

Работа с локальным репозиторием

- Команда init создает репозиторий в текущем каталоге
- Команда add добавляет измененные файлы в **stage**
- Команда rm помечает файл в **stage** как удаленный
- Команда reset сбрасывает изменения в текущем **stage**
- Команда commit сохраняет текущий **stage** в локальный репозиторий

Состояния файлов



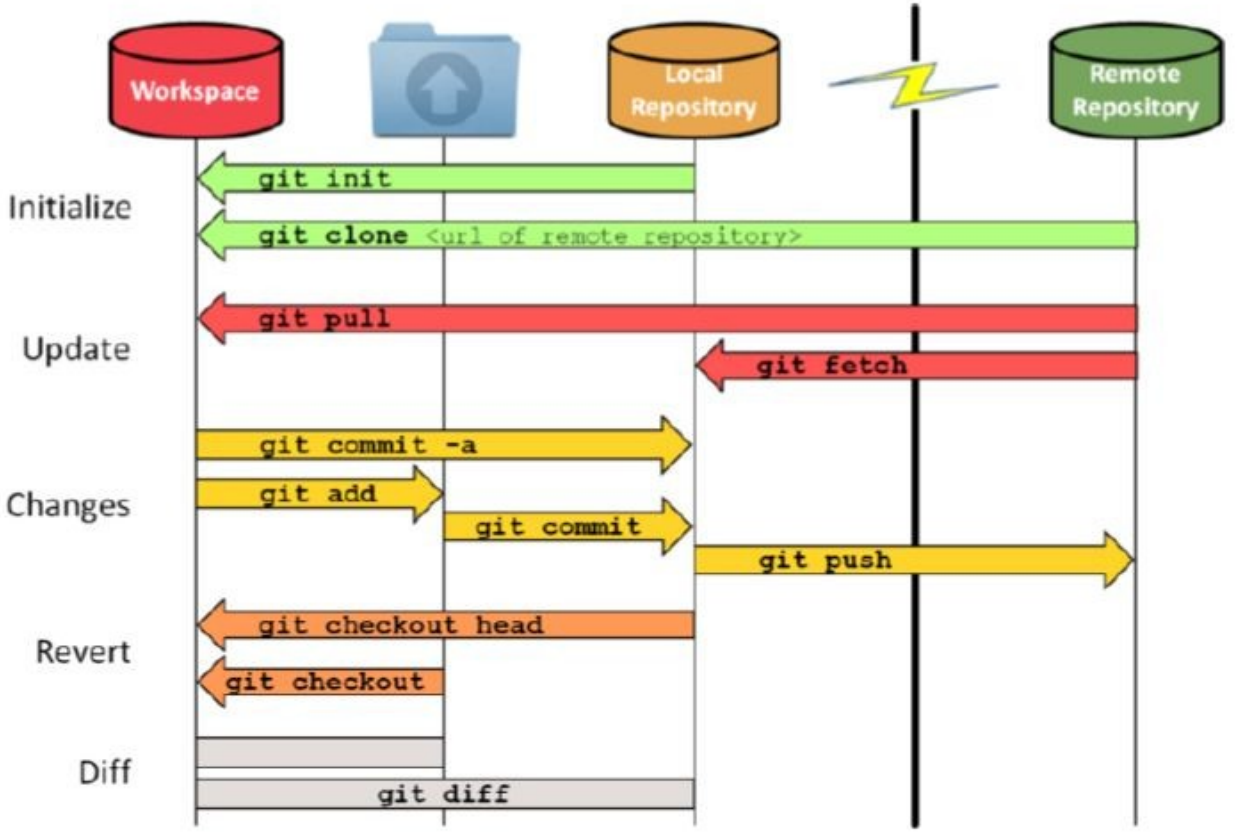
Определение состояний файлов

- Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда:
  - $ git status
- Увидеть изменения в более компактном виде:
  - $ git status -s

Работа с удаленным репозиторием

- Команда clone клонирует **репозиторий** и создаёт рабочую копию
- Команда push отправляет изменения в удаленный **репозиторий**
- Команда pull забирает изменения указанной ветки из удаленного **репозитория** и сливает их в текущую ветку
- Команда fetch забирает все изменения из удаленного **репозитория**

Жизненный цикл Git



Просмотр истории коммитов

- После того, как вы создали несколько коммитов или же склонировали репозиторий с уже существующей историей коммитов, вероятно вам понадобится возможность посмотреть что было сделано – историю коммитов:
  - $ git log
- Одним из самых полезных аргументов является -p или --patch, который показывает разницу (выводит патч), внесенную в каждый коммит
  - $ git log -p –2

Основные операции

- Были сделали изменения в файле test.cpp. Проверить статус:
  - $ git status
- Добавить изменения в **stage:**
  - $ git add test.cpp или $ git add .
- Сделать коммит в локальный репозиторий:
  - $ git commit -m “my first commit”
- Проверить изменения с удаленного репозитория и добавит в локальный:
  - $ git pull
- Добавить изменения в удаленный репозиторий:
  - $ git push