

# ТЕХНОЛОГИЯ И ПРОЦЕСС РАЗРАБОТКИ ПО (Л-6)



к.п.н., доцент Касаткин Д.  
А.

*e-mail: [kasatkinda@cfuv.ru](mailto:kasatkinda@cfuv.ru)*



# Система контроля версий

- Системы управления версиями (Version Control Systems, VCS) или Системы управления исходным кодом (Source Management Systems, SMS) — важный аспект разработки современного ПО.

## **VCS предоставляет следующие возможности:**

- Поддержка хранения файлов в репозитории.
- Поддержка истории версий файлов в репозитории.
- Нахождение конфликтов при изменении исходного кода и обеспечение синхронизации при работе в многопользовательской среде разработки.
- Отслеживание авторов изменений.



# Система контроля версий

- Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.



# Классификация :

- Централизованные/распределённые — в централизованных системах контроля версий вся работа производится с центральным репозиторием, в распределённых — у каждого разработчика есть локальная копия репозитория.
- Блокирующие/не блокирующие — блокирующие системы контроля версий позволяют наложить запрет на изменение файла, пока один из разработчиков работает над ним, в неблокирующих один файл может одновременно изменяться несколькими разработчиками.
- Для текстовых данных/для бинарных данных — для VCS для текстовых данных очень важна поддержка слияния изменений, для VCS с бинарными данными важна возможность блокировки



# Ежедневный цикл работы

Обычный цикл работы разработчика выглядит следующим образом:

## **Обновление рабочей копии.**

- Разработчик выполняет операцию обновления рабочей копии (update) насколько возможно

## **Модификация проекта.**

- Разработчик локально модифицирует проект, изменяя входящие в него файлы в рабочей копии.

## **Фиксация изменений.**

Завершив очередной этап работы над заданием, разработчик фиксирует (commit) свои изменения, передавая их на сервер. VCS может требовать от разработчика перед фиксацией выполнить обновление.



# Основные термины:

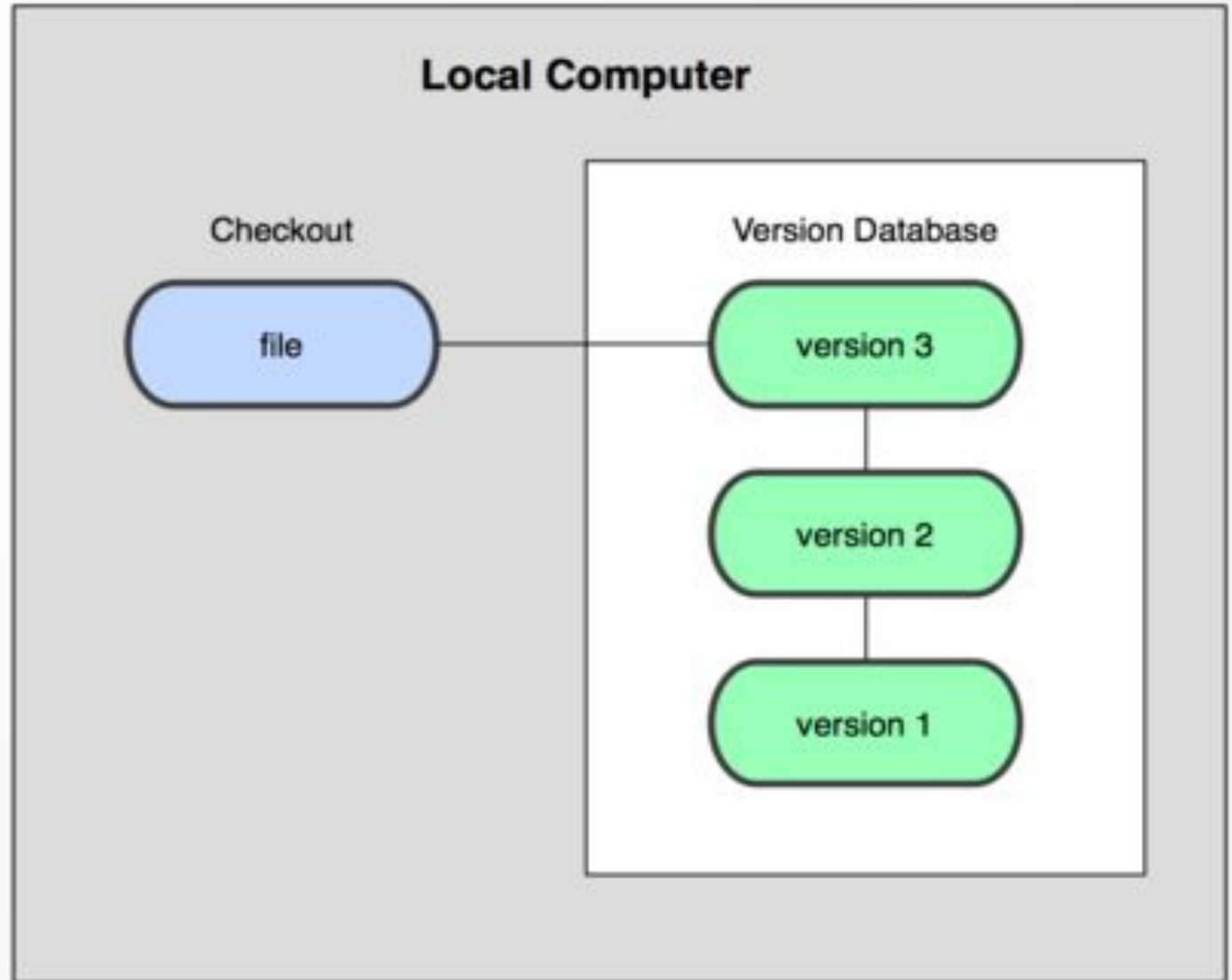
- **Working copy** – рабочая (локальная) копия документов.
- **repository, depot** — хранилище.
- **Revision** — версия документа. Новые изменения (changeset) создают новую ревизию репозитория.
- **check--in, commit, submit** - фиксация изменений.
- **check--out, clone** — извлечение документа из хранилища и создание рабочей копии.
- **Update, sync** — синхронизация рабочей копии до некоторого заданного состояния хранилища (в т. ч. и к более старому состоянию, чем текущее).
- **merge, integration** — слияние независимых изменений.
- **Conflict** — ситуация, когда несколько пользователей сделали изменения одного и того же участка документа.
- **head** — самая свежая версия (revision) в хранилище.  
**Origin** — имя главного сервера



# Ветвление

- **Ветвь (*branch*)** - направление разработки проекта, независимое от других
- Ветвь представляет собой копию части (как правило, одного каталога) хранилища, в которую можно вносить свои изменения, не влияющие на другие ветви.
- Документы в разных ветвях имеют одинаковую историю до точки ветвления и разные — после неё.
- Изменения из одной ветви можно переносить в другую.
- **Ствол (*trunk, mainline, master*)** — основная ветвь разработки проекта.

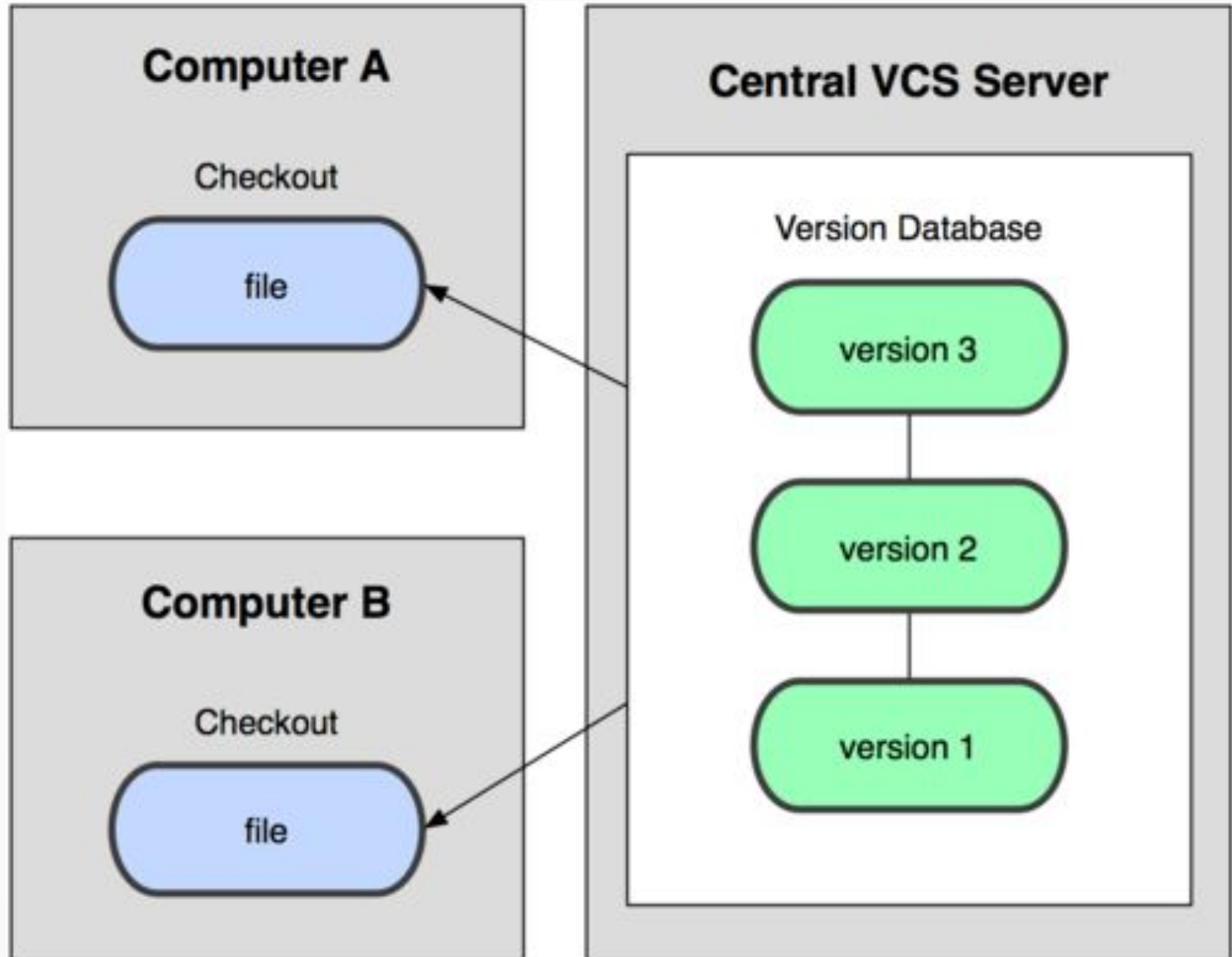
# Локальные системы контроля версий



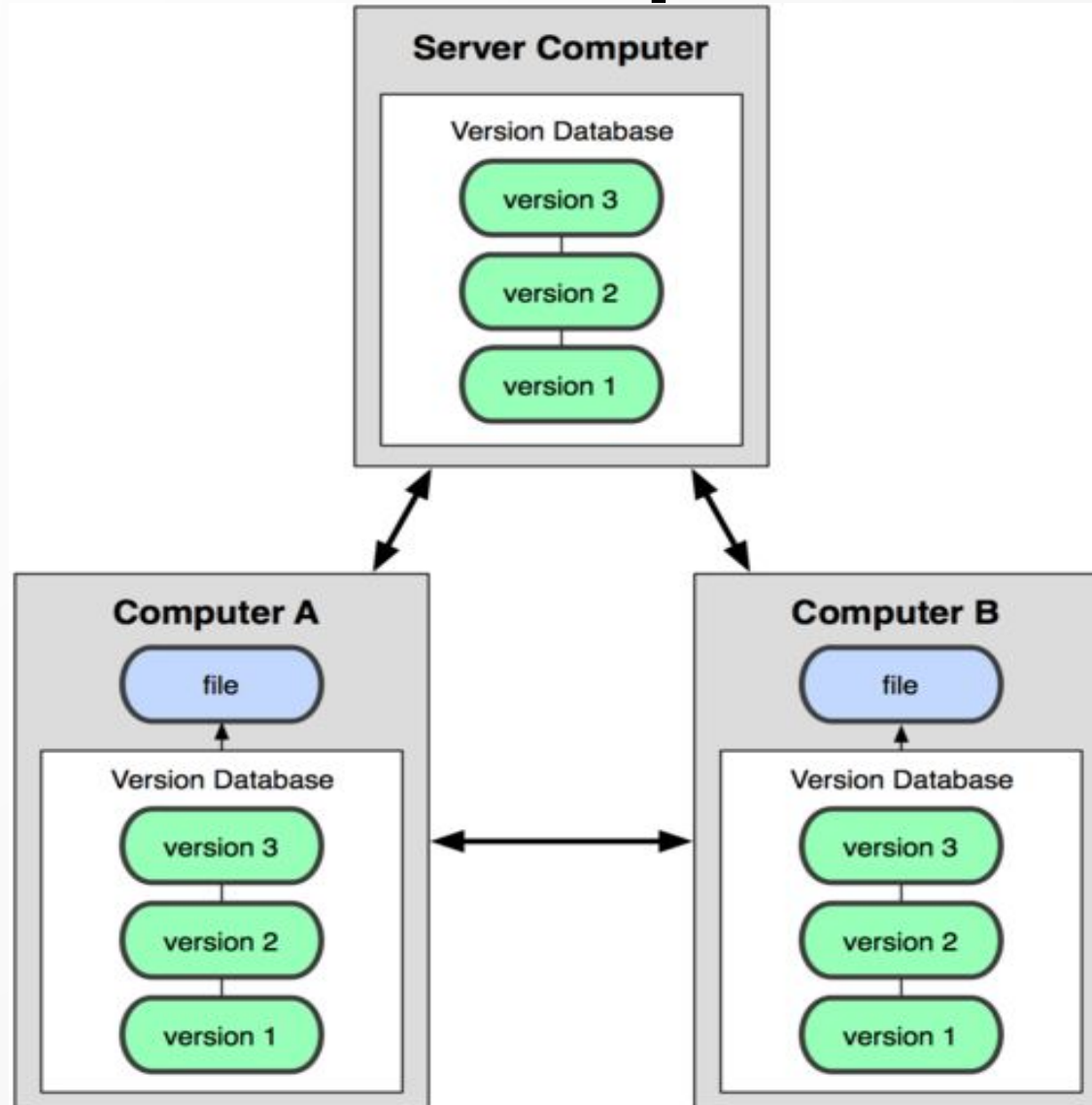




# Централизованные системы контроля версий



# Распределенные СИСТЕМЫ КОНТРОЛЯ Версий



# Краткое описание

## популярных

## распределенных СУВ



- Git (<http://git-scm.com/>) - распределенная система контроля версий, разработанная Линусом Торвальдсом. Изначально Git предназначалась для использования в процессе разработки ядра Linux, но позже стала использоваться и во многих других проектах — таких, как, например, X.org и Ruby on Rails, Drupal. На данный момент Git является самой быстрой распределенной системой, использующей самое компактное хранилище ревизий. Но в тоже время для пользователей, переходящих, например, с Subversion интерфейс Git может показаться сложным;
- Mercurial (<http://www.selenic.com/mercurial/>) - распределенная система, написанная на языке Python с несколькими расширениями на C. Из использующих Mercurial проектов можно назвать, такие, как, Mozilla и MoinMoin.
- Bazaar (<http://bazaar-vcs.org/>) - система разработка которой поддерживается компанией Canonical — известной своими дистрибутивом Ubuntu и сайтом <https://launchpad.net/>. Система в основном написана на языке Python и используется такими проектами, как, например, MySQL.
- Codeville (<http://codeville.org/>) - написанная на Python распределенная система использующая инновационный алгоритм объединения изменений (merge). Система используется, например, при разработке оригинального клиента BitTorrent.
- Darcs (<http://darcs.net/>) - распределенная система контроля версий



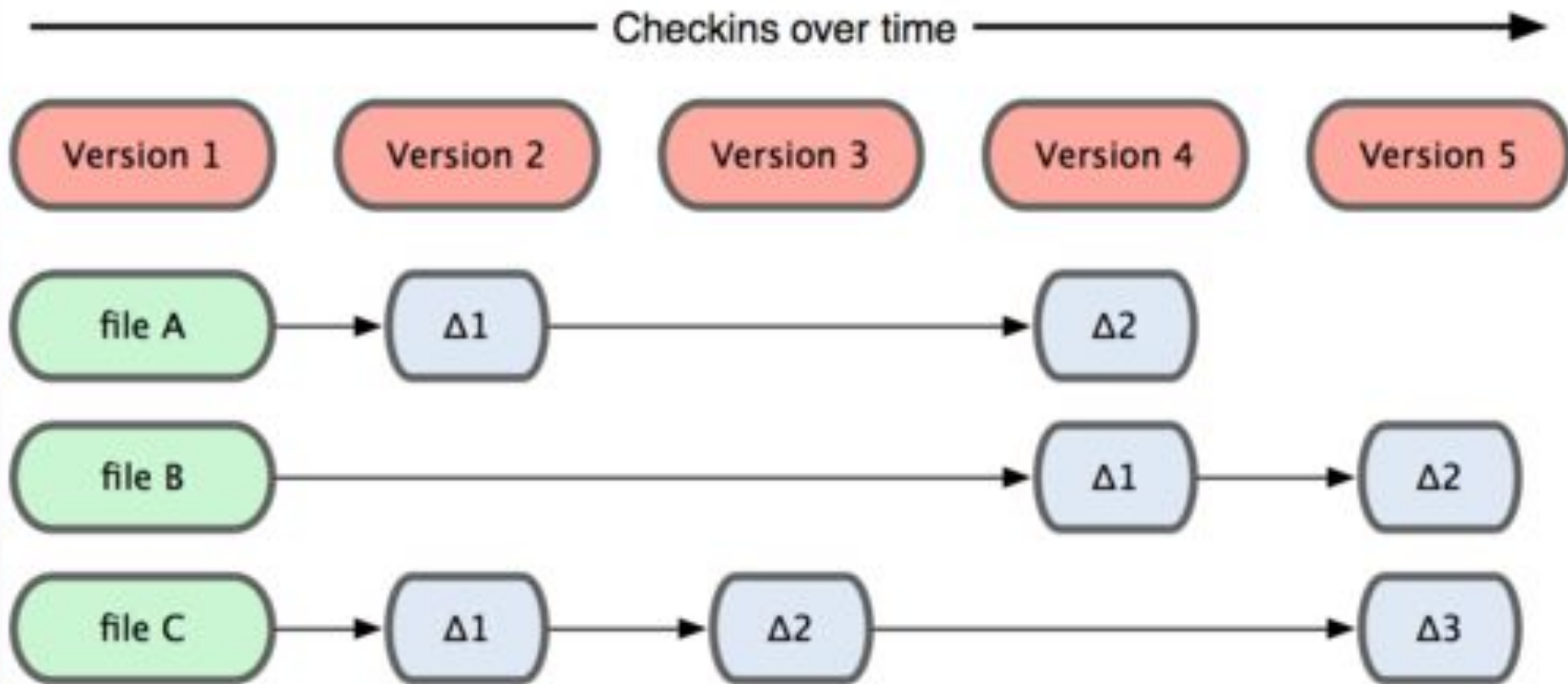
# Краткая история Git

Основные требования к новой системе были следующими:

- Скорость
- Простота дизайна
- Поддержка нелинейной разработки (тысячи параллельных веток)
- Полная распределённость
- Возможность эффективной работы с такими большими проектами, как ядро Linux (как по скорости, так и по размеру данных)

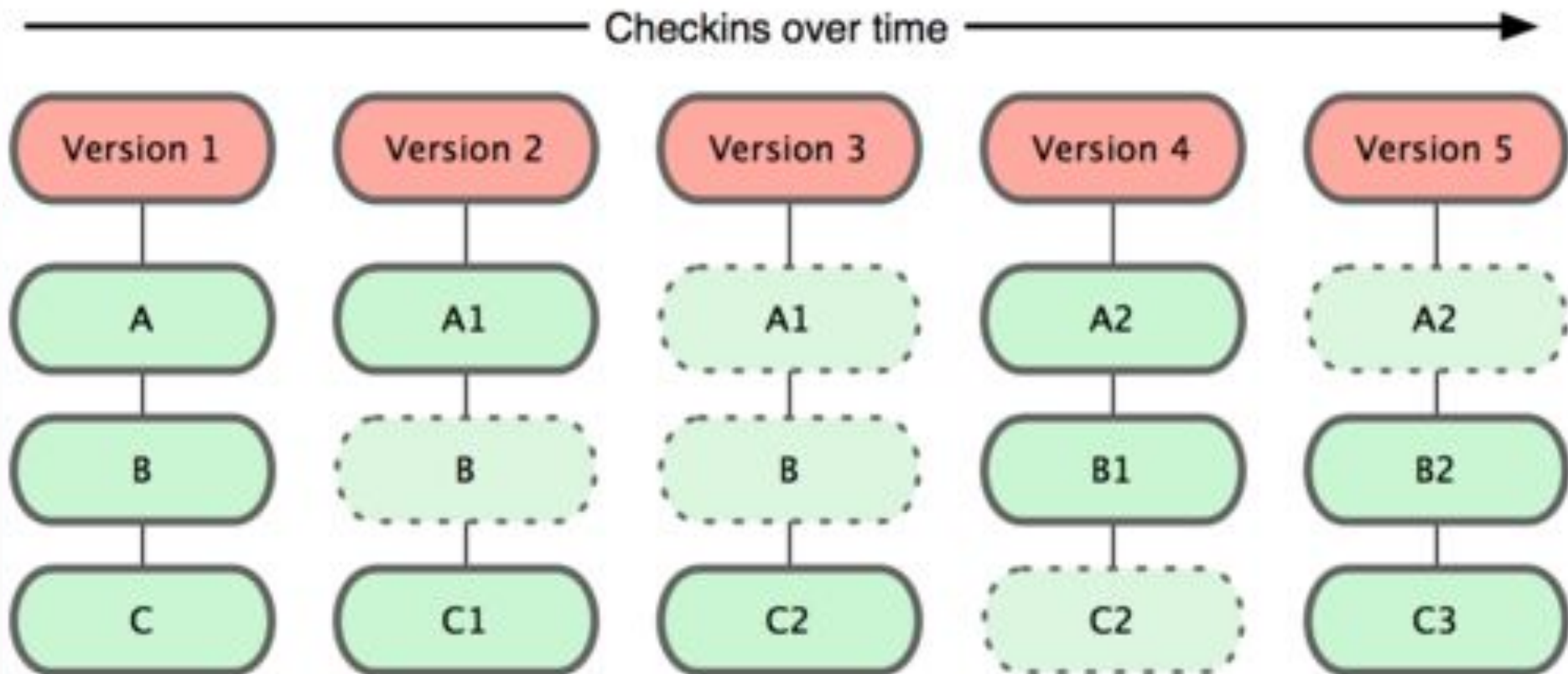


# Слепки вместо патчей





# Git хранит данные как слепки состояний проекта во времени





# Git следит за целостностью данных

- 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**

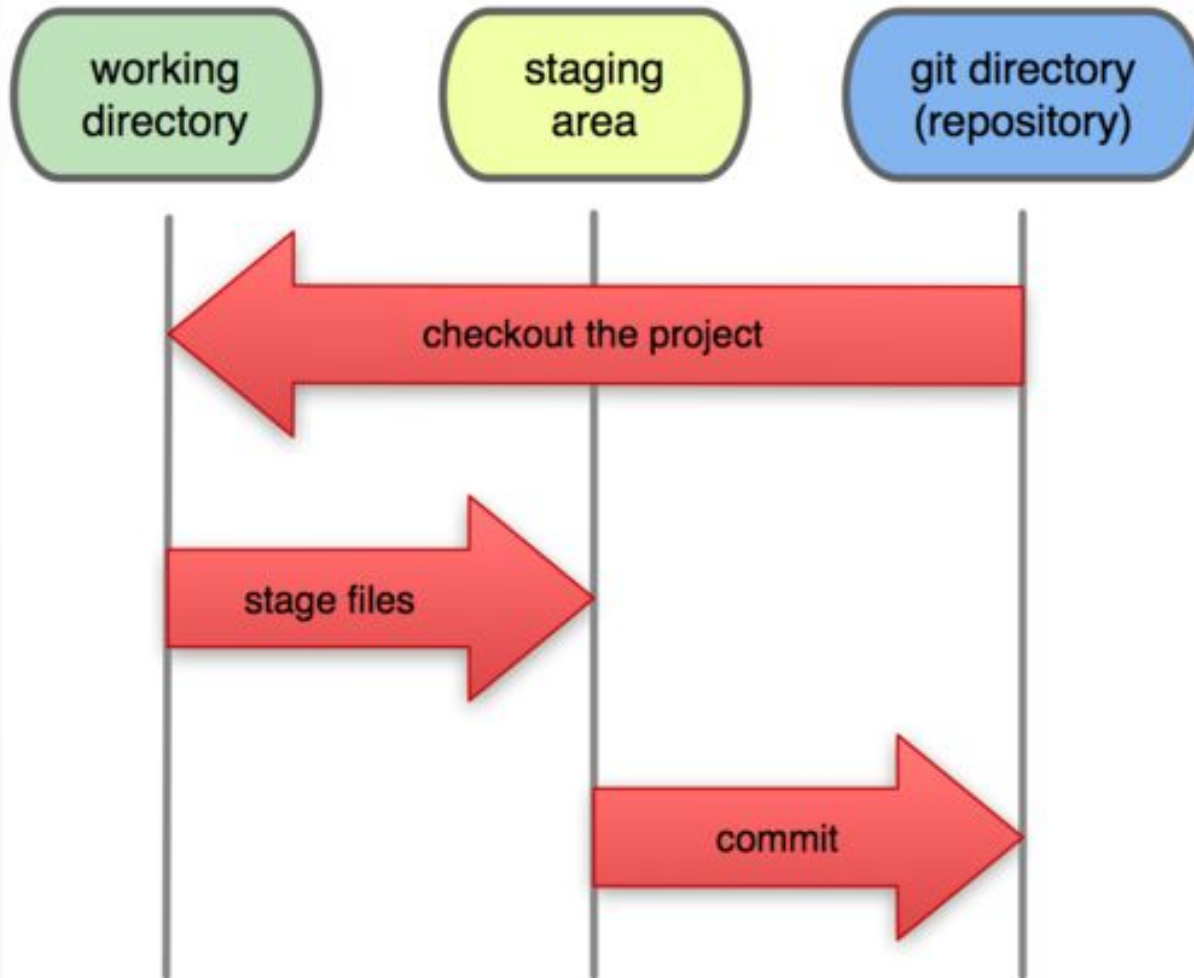
- Работая с Git'ом, вы будете встречать эти хеши повсюду, поскольку он их очень широко использует. Фактически, в своей базе данных Git сохраняет всё не по именам файлов, а по хешам их содержимого.





# Три состояния GIT

## Local Operations







# Установка GIT

- Установка в Linux

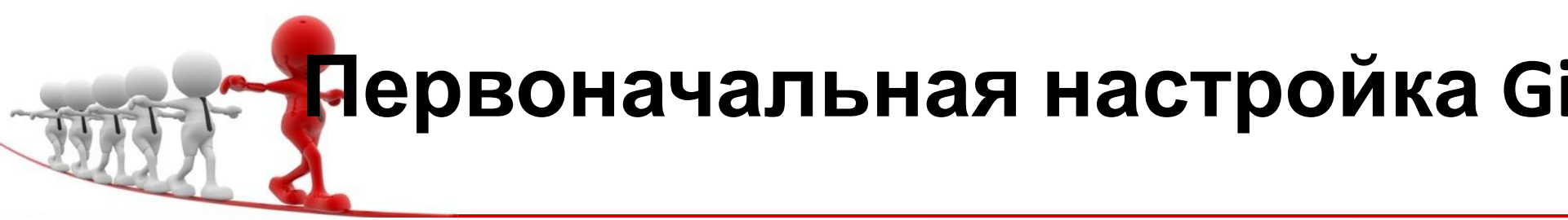
Если вы хотите установить Git под Linux как бинарный пакет, это можно сделать, используя обычный менеджер пакетов вашего дистрибутива. Если у вас Fedora, можно воспользоваться yum'ом:

- **`$ yum install git-core`**

Если же у вас дистрибутив, основанный на Debian, например, Ubuntu, попробуйте apt-get: **`$ apt-get install git`**

- Установка на Mac

Есть два простых способа установить Git на Mac. Самый простой — использовать графический инсталлятор Git'a, который вы можете скачать со страницы на [Google Code](#):



# Первоначальная настройка Git

- В состав Git'a входит утилита `git config`, которая позволяет просматривать и устанавливать параметры, контролирующие все аспекты работы Git'a и его внешний вид. Эти параметры могут быть сохранены в трёх местах:
  - Файл `/etc/gitconfig` содержит значения, общие для всех пользователей системы и для всех их репозиториев. Если при запуске `git config` указать параметр `--system`, то параметры будут читаться и сохраняться именно в этот файл.
  - Файл `~/.gitconfig` хранит настройки конкретного пользователя. Этот файл используется при указании параметра `--global`.
  - Конфигурационный файл в каталоге Git'a (`.git/config`) в том репозитории, где вы находитесь в данный момент. Эти параметры действуют только для данного конкретного репозитория. Настройки на каждом следующем уровне подменяют настройки из предыдущих уровней, то есть значения в `.git/config` перекрывают соответствующие значения в `/etc/gitconfig`.



# Имя пользователя

- Первое, что вам следует сделать после установки Git'a, — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git'e содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена:
- `$ git config --global user.name "Denis Kasatkin"`
- `$ git config --global user.email d.kasatkin@outlook.com`



# Выбор редактора

Вы указали своё имя, и теперь можно выбрать текстовый редактор, который будет использоваться, если будет нужно набрать сообщение в Git'e. По умолчанию Git использует стандартный редактор вашей системы, обычно это Vi или Vim. Если вы хотите использовать другой текстовый редактор, например, Emacs, можно сделать следующее:

```
$ git config --global core.editor emacs
```



# Утилита сравнения

- Другая полезная настройка, которая может понадобиться — встроенная diff-утилита, которая будет использоваться для разрешения конфликтов слияния. Например, если вы хотите использовать vimdiff:
- `$ git config --global merge.tool vimdiff`
- Git умеет делать слияния при помощи kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, esmerge и opendiff, но вы можете настроить и другую утилиту.



# Проверка настроек

- Если вы хотите проверить используемые настройки, можете использовать команду `git config --list`, чтобы показать все, которые Git найдёт:
- `$ git config --list`
- `user.name=Denis Kasatkin`
- `user.email=d.kasatkin@outlook.com`
- `color.status=auto`
- `color.branch=auto`
- `color.interactive=auto`
- `color.diff=auto`
- ...



# Как получить помощь?

- Если вам нужна помощь при использовании Git'a, есть три способа открыть страницу руководства по любой команде Git'a:
- `$ git help <команда>` `$ git <команда> --help`  
`$ man git-<команда>`
- Например, так можно открыть руководство по команде `config`:
- `$ git help config`



# Создание Git-репозитория


- Для создания Git-репозитория существуют два основных подхода. Первый подход — импорт в Git уже существующего проекта или каталога. Второй — клонирование уже существующего репозитория с сервера.
- **Создание репозитория в существующем каталоге**
- Если вы собираетесь начать использовать Git для существующего проекта, то вам необходимо перейти в проектный каталог и в командной строке ввести
- `$ git init`





# Версионный контроль

- Если вы хотите добавить под версионный контроль существующие файлы (в отличие от пустого каталога), вам стоит проиндексировать эти файлы и осуществить первую фиксацию изменений. Осуществить это вы можете с помощью нескольких команд `git add` указывающих индексировать файлы, а затем `commit`:
- `$ git add *.c`
- `$ git add README`
- `$ git commit -m 'initial project version'`
- Мы разберём, что делают эти команды чуть позже. На данном этапе, у вас есть Git-репозиторий с добавленными файлами и начальным коммитом.



# Клонирование существующего репозитория

- Клонирование репозитория осуществляется командой `git clone [url]`. Например, если вы хотите клонировать библиотеку Ruby Git, известную как Grit, вы можете сделать это следующим образом:
- `$ git clone git://github.com/libgit2/rugged.git`
- Эта команда создаёт каталог с именем `rugged`, инициализирует в нём каталог `.git`, скачивает все данные для этого репозитория и создаёт (`checks out`) рабочую копию последней версии. Если вы зайдёте в новый каталог `rugged`, вы увидите в нём проектные файлы, пригодные для работы и использования.



# Клонирование существующего репозитория

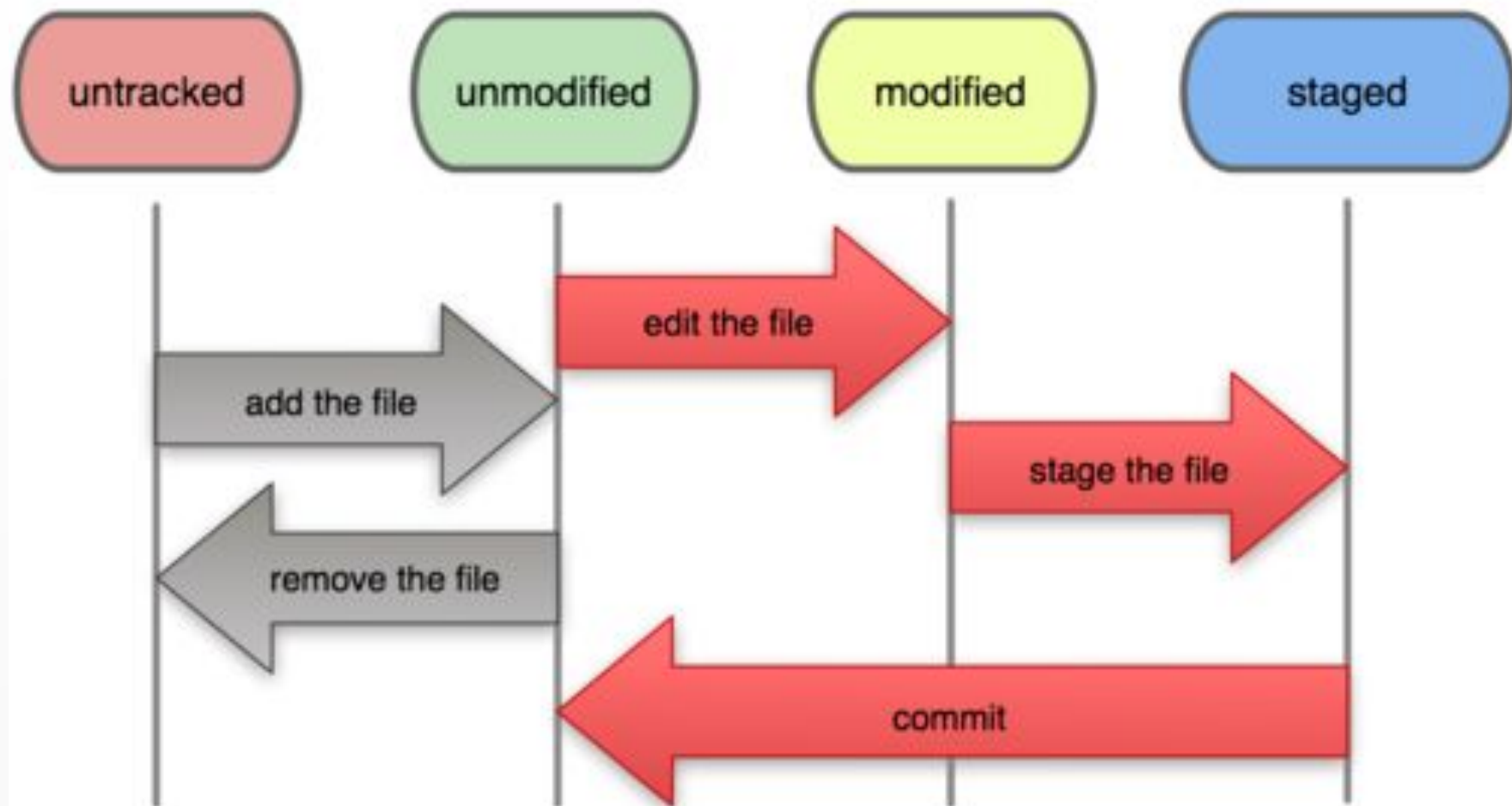
- Если вы хотите клонировать репозиторий в каталог, отличный от `rugged`, можно это указать в следующем параметре командной строки:

```
$ git clone git://github.com/libgit2/rugged.git  
myrugged
```



# Запись изменений в репозитории

## File Status Lifecycle





# Определение состояния файлов

- Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда `git status`. Если вы выполните эту команду сразу после клонирования, вы увидите что-то вроде этого:
- `$ git status # On branch master nothing to commit (working directory clean)`
- Это означает, что у вас чистый рабочий каталог, другими словами — в нём нет отслеживаемых изменённых файлов. Git также не обнаружил неотслеживаемых файлов, в противном случае они бы были перечислены здесь. И наконец, команда сообщает вам на какой ветке (branch) вы сейчас находитесь. Пока что это всегда ветка `master` — это ветка по умолчанию. Позже подробнее поговорим о



# Графические интерфейсы

- [GitKraken](#) — кроссплатформенный бесплатный клиент Git.
- [SmartGit](#) — кроссплатформенный интерфейс для Git на Java.
- [gitk](#) — простая и быстрая программа, написана на [Tcl/Tk](#), распространяется с самим Git.
- [QGIt](#), интерфейс которого написан с использованием [Qt](#), во многом схож с [gitk](#), но несколько отличается набором возможностей. В настоящее время существуют реализации на Qt3 и Qt4.
- [Giggle](#) — вариант на [Gtk+](#).
- [gitg](#) — ещё один интерфейс для [gtk+/GNOME](#)
- [Git Extensions](#) — кроссплатформенный вариант на [.NET](#).
- [TortoiseGit](#) — интерфейс, реализованный как расширение для [проводника Windows](#).
- [SourceTree](#) — бесплатный git клиент для Windows и Mac.
- [Git-cola](#) — кроссплатформенный интерфейс на [Python](#).
- [GitX](#) — оболочка для [Mac OS X](#) с интерфейсом [Cocoa](#), интерфейс схож с [gitk](#).
- [Gitti](#) — оболочка для [Mac OS X](#) с интерфейсом [Cocoa](#).
- [Gitbox](#) — оболочка для [Mac OS X](#) с интерфейсом [Cocoa](#).
- [Github](#)-клиент
- [StGit](#) — написанная на [Python](#) система управления коллекцией патчей (Catalin Marinas)
- [GitTower](#) — коммерческий клиент Git для Mac и Windows







