РХТУ им. Д.И. Менделеева
Кафедра информационных компьютерных технологий

Теория проектирования информационных систем

Лекция 9: Система управления версиями git. Первые команды

Преподаватель: к.т.н., доцент кафедры ИКТ

Митричев Иван Игоревич

В лекции использованы материалы онлайн-учебника
https://git-scm.com/book/ru/v2/

Системы контроля версий

Система управления версиями (СУВ) – программа (или комплекс программ), позволяющая сохранять различные версии документов (обычно – программного кода), объединять изменения различных авторов и в любое время возвращаться к любой версии документов.

Цикл работы с СУВ:

1) Начальное получение проекта – выполняется однократно (git clone)

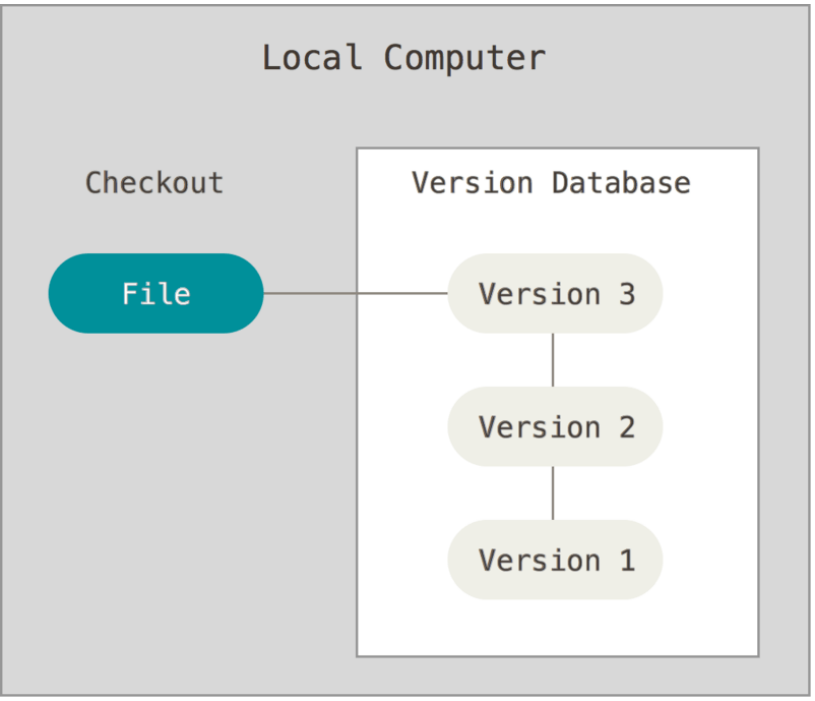
Цикл: 2) ежедневное обновление рабочей копии проекта (git pull = git fetch + git merge). Разрешение конфликтов

3) разработка (изменение рабочей копии)

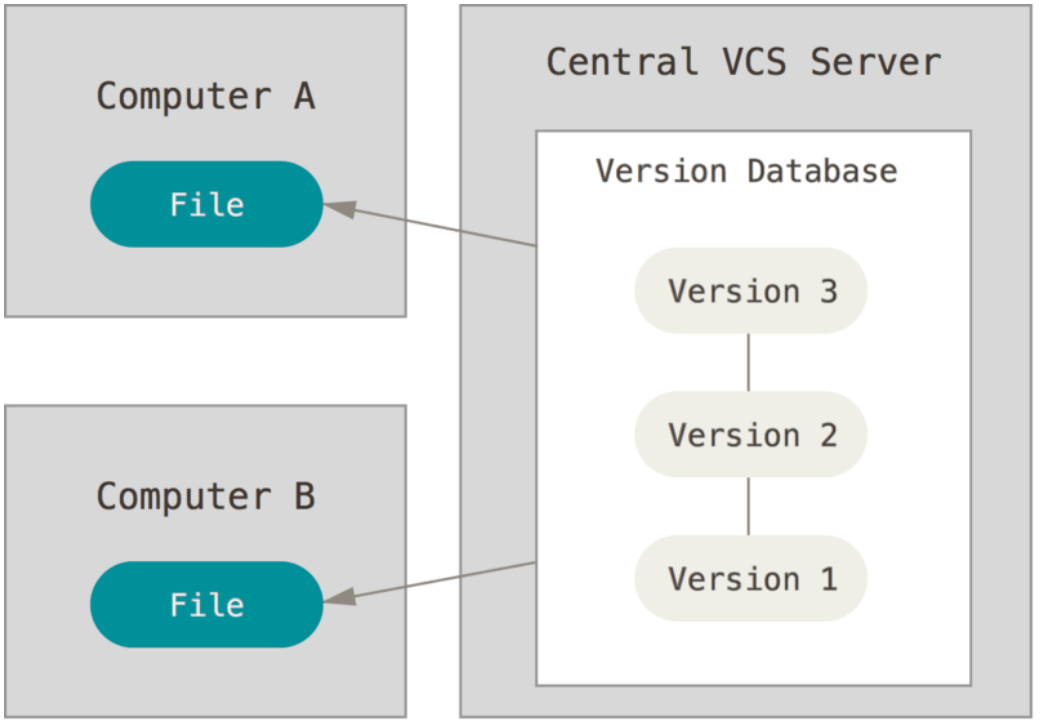
4) сохранение изменений в рабочей копии (фиксация)

5) отправка изменений в репозиторий. Разрешение конфликтов администратором репозитория

Какие бывают СУВ – 1



Локальные (один разработчик)

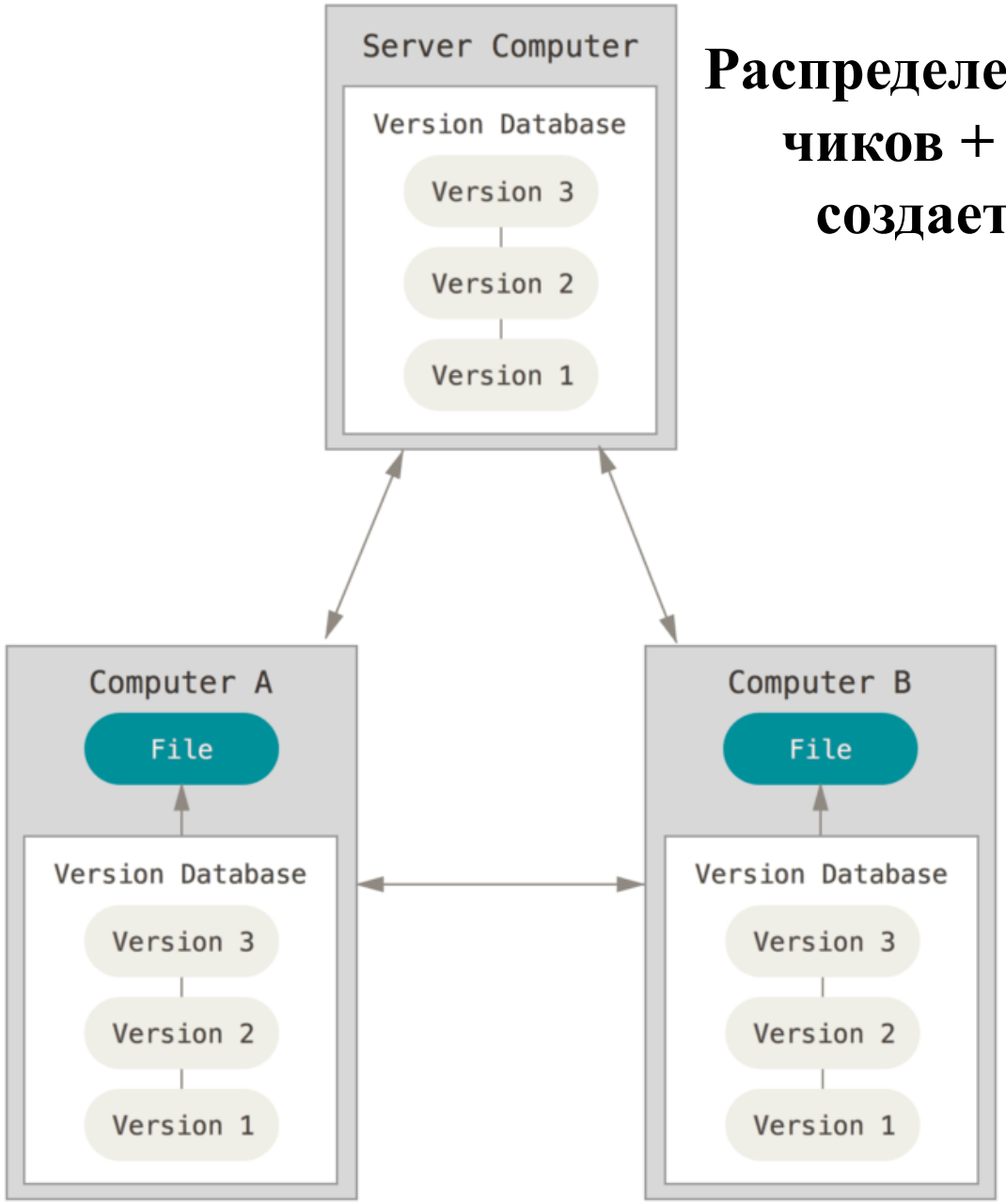


Централизованные (несколько разработчиков)

Минусы:

Вся информация хранится на локальном или удаленном сервере. При сбое сервера или ошибке доступа к нему работа невозможна. Нужно резервное копирование сервера, создание альтернативной подсистемы доступа к нему.

Какие бывают СУВ – 2



Распределенные (несколько разработчиков + у каждого разработчика создается копия репозитория)

Ответственность за синхронизацию локального и удаленного репозиториев лежит на разработчике. Без синхронизации процесс разработки нарушает принципы распределенных СУВ и становится неэффективен/невозможен.

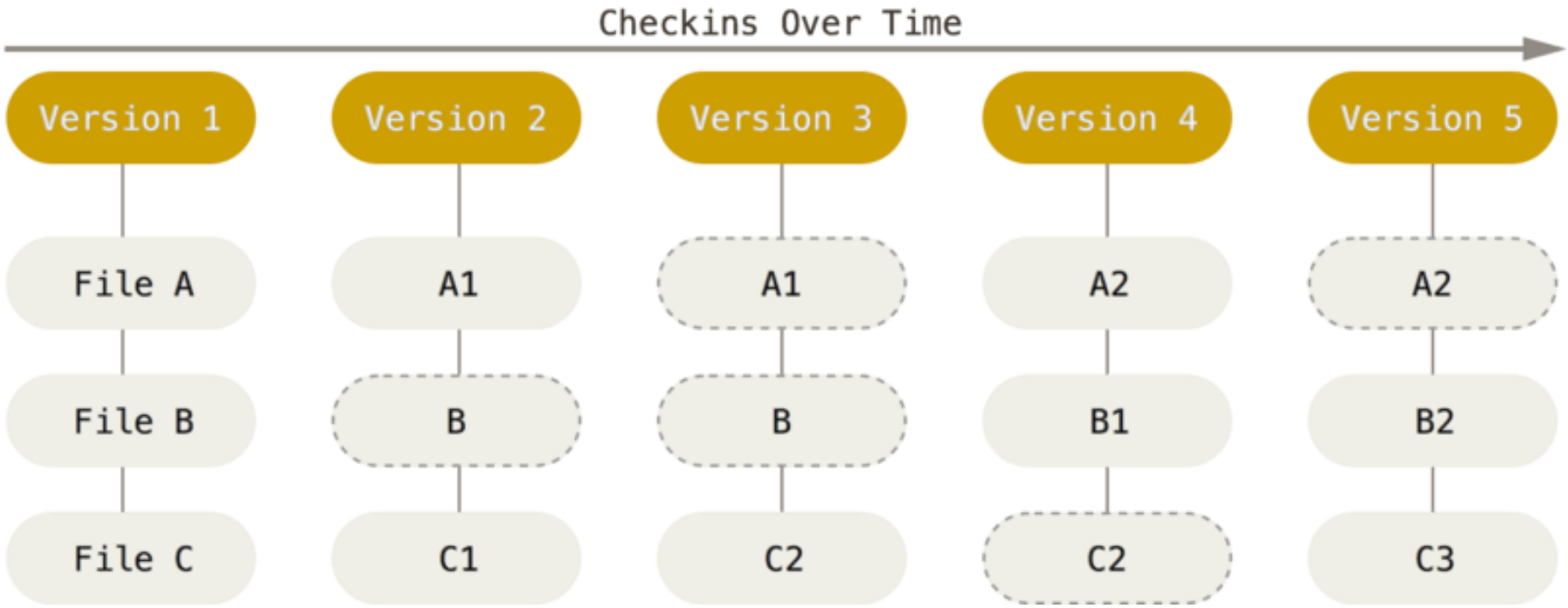
Git, Mercurial, Bazaar

Общие правила разработки с СУВ

1) Каждый проект имеет основную ветвь, в которую изменения может вносить только ограниченный круг лиц (главные разработчики, администраторы). Может быть несколько основных ветвей для версий программы, коренным образом отличающихся друг от друга (полностью переписан код).
2) Главная ветвь проекта содержит только проверенные изменения, проект в ней всегда собирается.
3) Конечная сборка осуществляется для версий кода из репозитория. Для этого применяют автоматическую сборку на чистой системе (виртуальная машина) с установкой всех зависимостей с нуля.
4) Для значимых изменений (feature) необходимо создать отдельную ветвь кода, а после окончания работы – объединить ее с родительской.

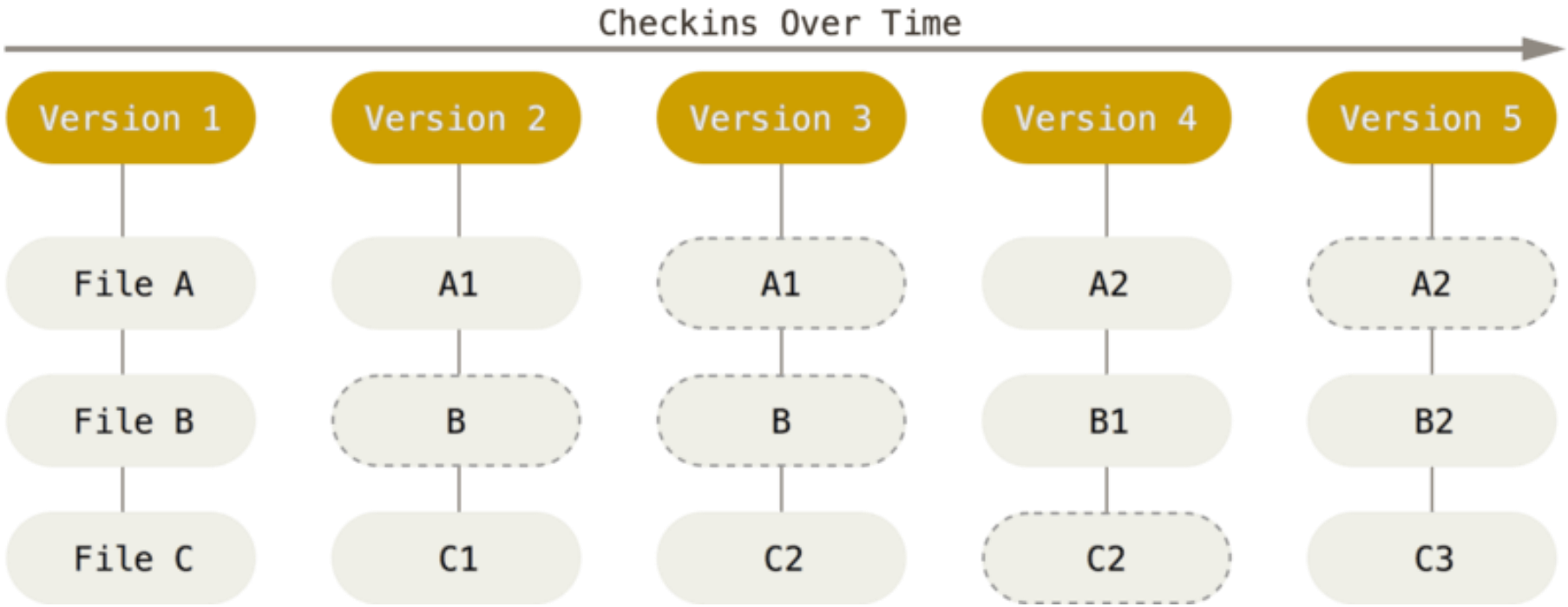
SVN

Хранит изменения, произведенные в каждом файле (дельты). Иногда сохраняет полностью состояние системы (снимок), чтобы быстрее получать любую версию.



Git

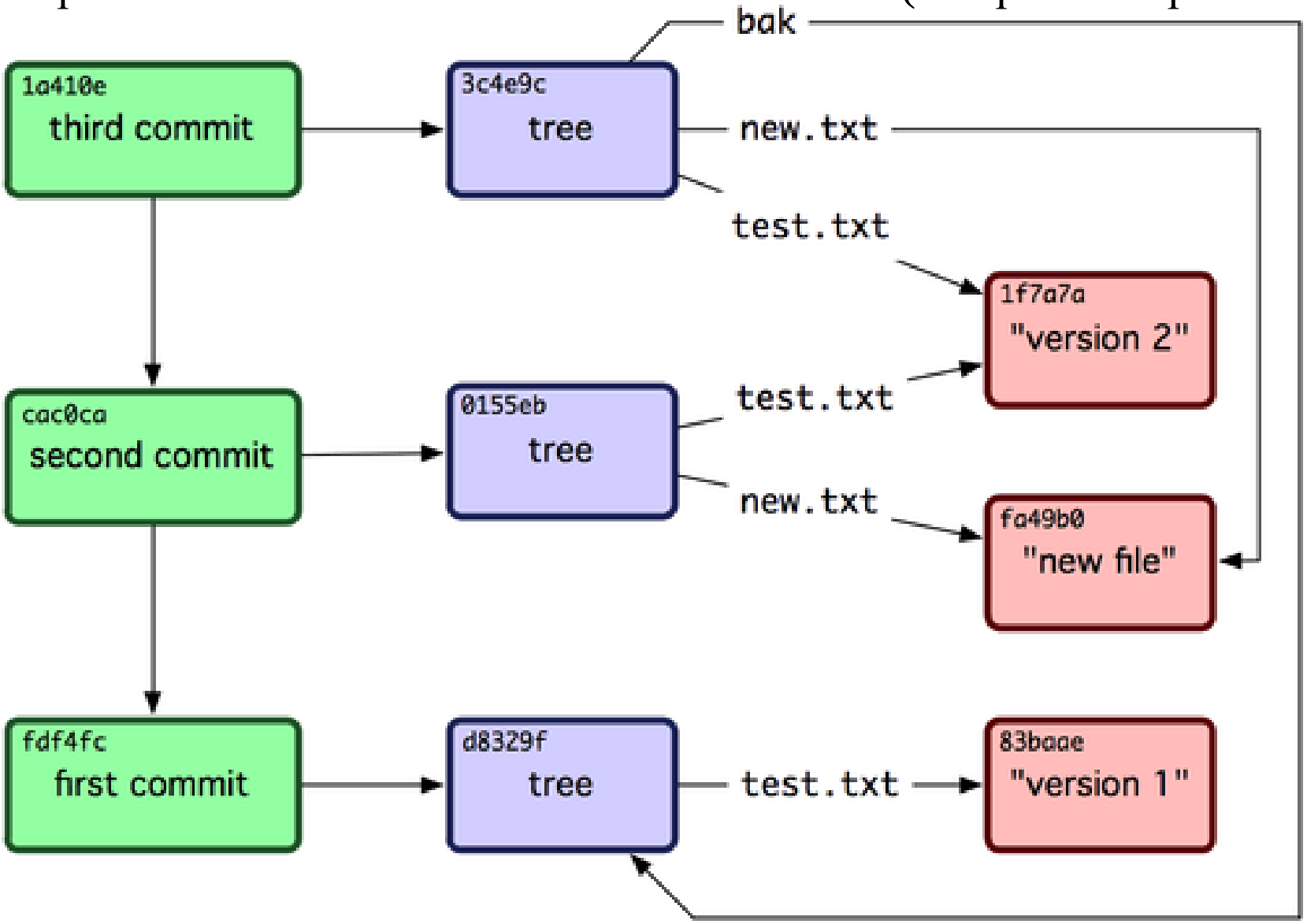
Для разработки ядра Linux Линус Торвальдс с соавторами в начале 2000х использовал проприетарную СУВ. С 2005 года проект перешел на собственную СУВ – git.
Хранит полный снимок репозитория после каждого изменения.



+ быстрота
+ распределенная
+ локальная копия репозитория (не обязательна сеть)

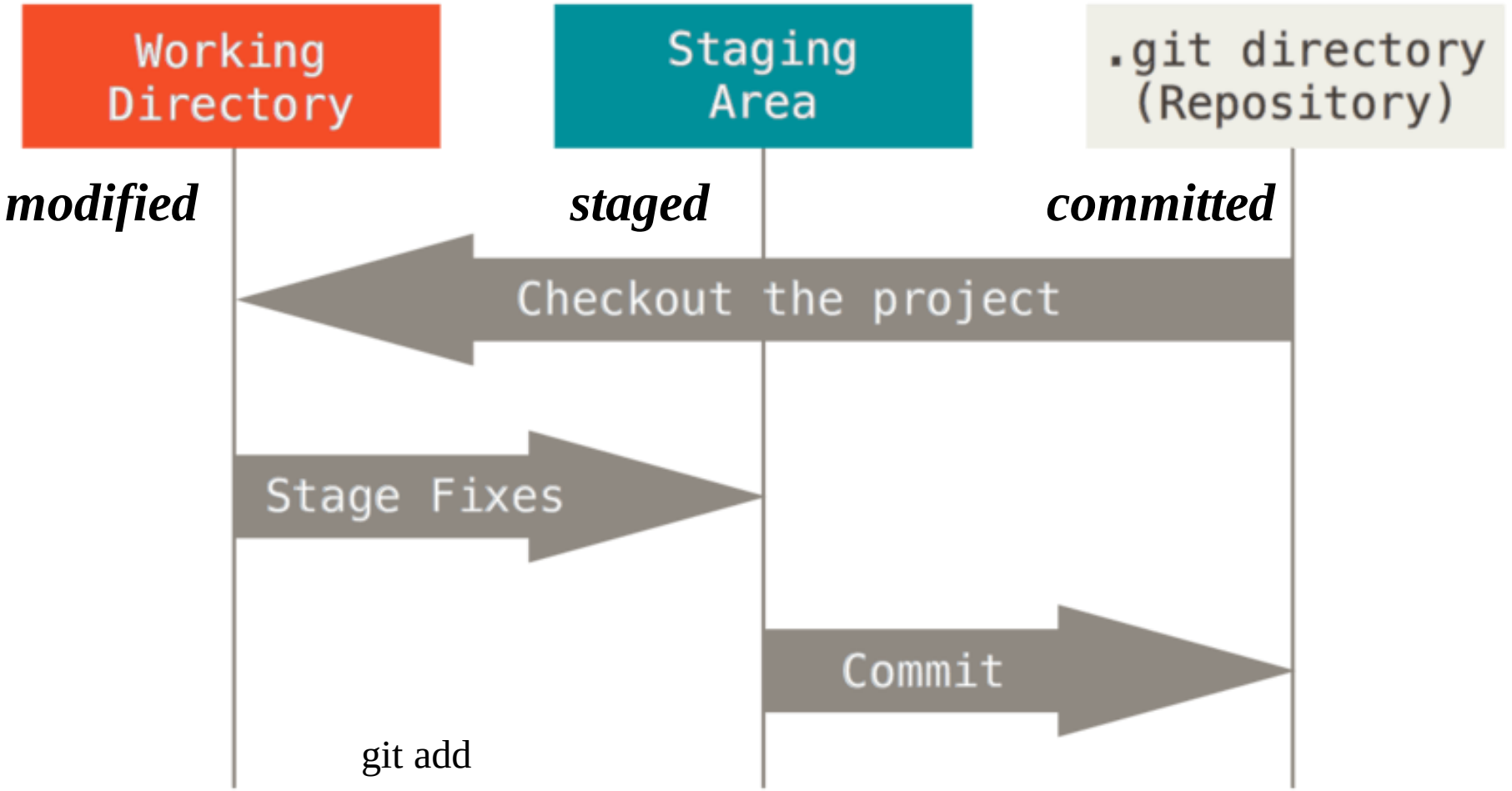
Хранение информации в Git

Три типа объектов: коммиты, деревья и блобы. Имеют хэш SHA-1.
Короткий hash коммита называют commit-ID (6 первых 16-ричн. чисел)



Хранение информации в Git

Рабочая копия (директория) Сцена (индекс) Локальный репозиторий



Индекс – то, что попадет в следующий коммит

Установка Git

https://git-scm.com/book/ru/v2/Введение-установка-Git

Linux:
sudo apt install git
или
sudo yum install git

Mac:
http://git-scm.com/download/mac

Windows:
http://git-scm.com/download/win
Можно вместо этого использовать инструменты Github:
http://windows.github.com/

Настройка Git

Файлы настроек

1. /etc/gitconfig (--system) – для всех пользователей (Windows – внутри каталого MSys)
2. ~/.gitconfig , также ~/.config/git/config (--global) – для текущего пользователя (Windows - $HOME\.gitconfig)
3. .git/config – для текущего репозитория

git config --global user.name “Vasya Pupkin"
git config --global user.email vpupkin@somemail.com
Linux: git config --global core.editor nano
Windows: git config --global core.editor '"C:\Windows\System32\notepad.exe"'

Создание репозитория

1) «С нуля»
git init – инициализировать репозиторий в текущей директории

2) Клонирование существующего репозитория (создание локальной копии)
git clone https://github.com/imitrichev/CPP2018.git

git add – добавить файлы
git add *.h *.cpp
git commit -m 'Здесь могла быть ваша реклама'

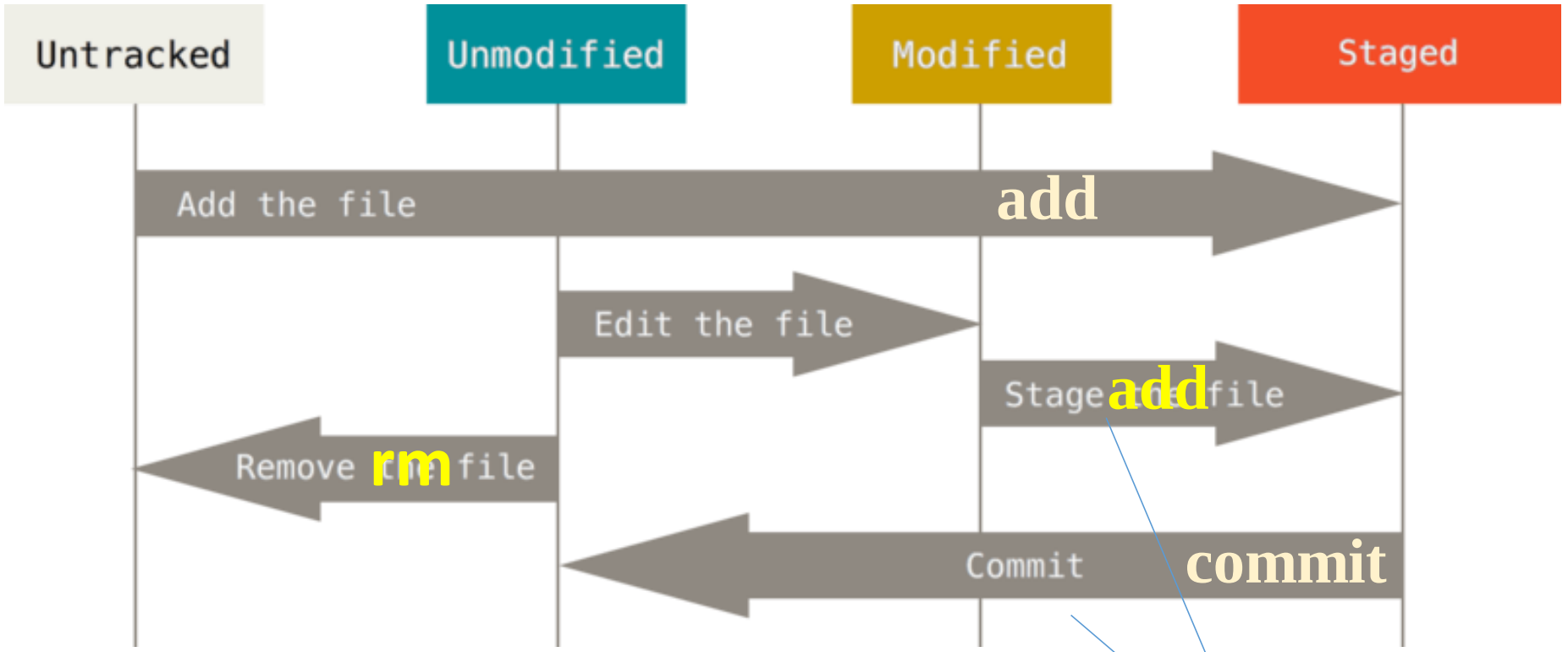
Жизненный цикл файлов

неотсле-жива-емый

неизме-ненный

изме-ненный

подготов-ленный к коммиту



git status

commit -a

stage/unstage

git add файл – добавить в индекс. Если после добавления файлов в индекс внести изменения в эти файлы, а потом сделать commit, то в репозитории сохранится добавленная в индекс версия файла, а изменения так и останутся непро-индексированными.

git rm -f файл - удалить подготовленный к коммиту файл из индекса, в т.ч., из рабочей копии. Файл становится неотслеживаемый. Без опции -f проиндексированный файл не удалится. Опасная опция - изменения, не закрепленный commit, теряются навсегда!

git rm --cached - удалить из индекса, оставить на диске.

Восстановление файла

git checkout -- файл - восстановить отдельный файл из локальной копии репозитория. Без подтверждения.

git checkout коммит/ветка - восстановить код из коммита/ветки. Проверяет, есть ли несохраненные изменения, требует сохранения.

git reset HEAD файл - **отменить индексирование**, файл остается в состоянии modified (то же, что с опцией --mixed).

git reset --hard HEAD файл - отменить индексирование и **перезаписать** файл из коммита HEAD.

git reset --soft HEAD~ - переместить указатель ветки на 1 коммит назад, не сбрасывать индексирование файлов.

git reset --hard HEAD~ отменить индексирование, **перезаписать без подтверждения все файлы** из предыдущего коммита.

Поправить коммит

git commit --amend - поправить последний коммит (будут учтены все изменения файлов, проиндексированные на текущие момент и внесены как поправки в последний коммит).

Спрятать на время

git stash - сохранить «в заначку» непроиндексированные изменения;

git stash pop - вернуть состояние файлов «из заначки».

ПРИНЦИП:
Чаще делайте commit! Любое изменение, которое было закоммичено когда-либо, восстановимо (если не применялись специальные алгоритмы удаления конфиденциальной информации - удаление ссылок и объектов). Незакоммиченные изменения теряются навсегда (checkout, reset --hard или --amend).

.gitignore

Игнорировать файлы (git add) – файл .gitignore

* - соответствует 0 или более символам;
[abc] — любой из символов, указанных в скобках.
Можно указывать диапазон: [a-c]
? - соответствует одному символу;
! – обращение шаблона; / в начале - только в текущей папке;
/ в конце - только папки; ** - все вложенные папки.

Пример файла .gitignore

```
# комментарий. Игнорировать все .a
*.a
# но не игнорировать lib.a
!lib.a
# игнорировать файл TODO в корневой папке, однако не
игнорировать вложенные файлы, к примеру subdir/TODO
/TODO
# игнорировать все файлы в директории build
build/
# игнорировать doc/notes.txt, но не doc/server/arch.txt
doc/*.txt
# игнорировать все pdf в doc и во вложенных папках
doc/**/*.pdf
```