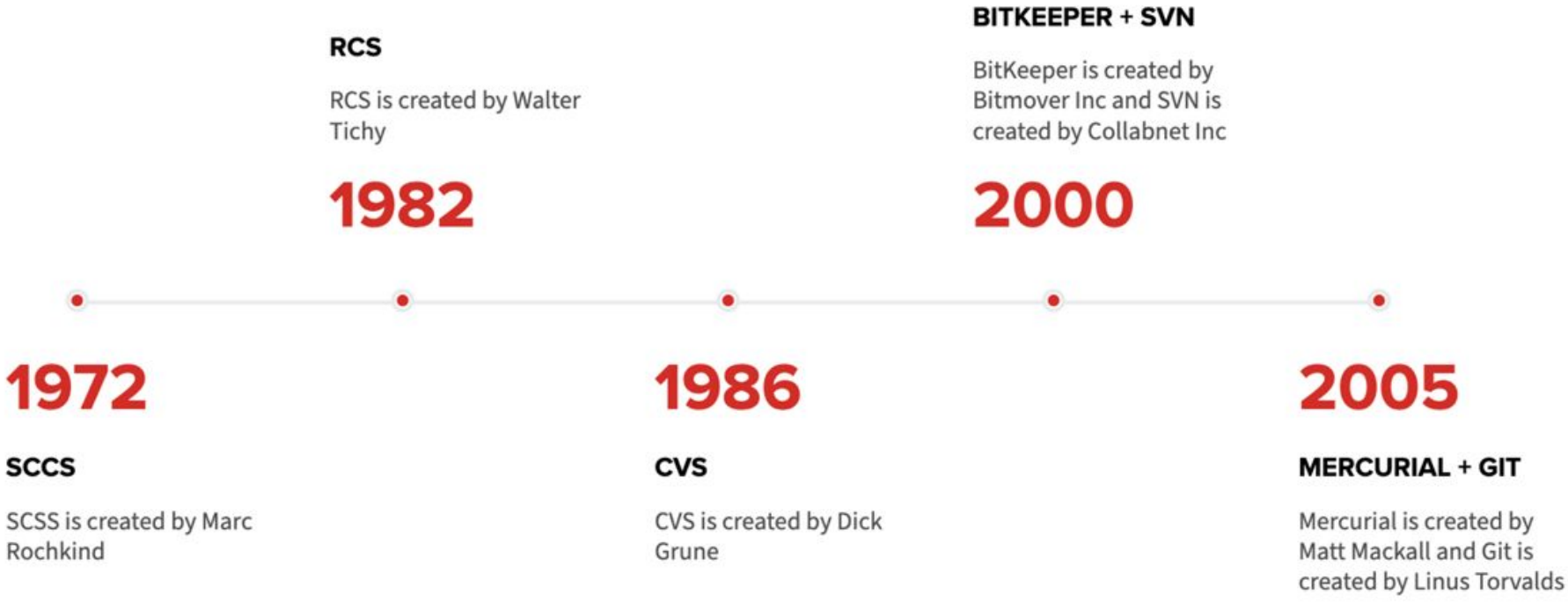
Version control

Хронология


1972
SCCS
SCSS is created by Marc Rochkind
1982
RCS
RCS is created by Walter Tichy
1986
CVS
CVS is created by Dick Grune
2000
BITKEEPER + SVN
BitKeeper is created by Bitmover Inc and SVN is created by Collabnet Inc
2005
MERCURIAL + GIT
Mercurial is created by Matt Mackall and Git is created by Linus Torvalds

Первое поколение

Системы контроля версий (VCS) первого поколения отслеживали изменения в отдельных файлах, а редактирование поддерживалось только локально и одним пользователем за раз.

SCCS (Source Code Control System) - считается одной из первых успешных систем управления версиями. Она была разработана в 1972 году Марком Рочкиндом из Bell Labs. Система написана на C и создана для отслеживания версий исходного файла.

RCS (Revision Control System) - написана в 1982 году Уолтером Тихи на языке C в качестве альтернативы системе SCCS, которая в то время не была опенсорсной.

Второе поколение

В VCS второго поколения появилась поддержка сети, что привело к централизованным хранилищам с «официальными» версиями проектов.

CVS (Concurrent Versions System) - создана Диком Груном в 1986 году с целью добавить в систему управления версиями поддержку сети. Впервые CVS позволил нескольким разработчикам одновременно работать с одними и теми же файлами. Это реализовано с помощью модели централизованного репозитория.

SVN (Subversion) - создана в 2000 году компанией Collabnet Inc., а в настоящее время поддерживается Apache Software Foundation. Система написана на C и разработана как более надёжное централизованное решение, чем CVS. Как и CVS, Subversion использует модель централизованного репозитория. Удалённым

Третье поколение

Систему Git разработал в 2005 году Линус Торвальдс (создатель Linux). Она написана в основном на C в сочетании с некоторыми сценариями командной строки. Отличается от VCS по функциям, гибкости и скорости. Торвальдс изначально написал систему для кодовой базы Linux, но со временем её сфера использования расширилась, и сегодня это самая популярная в мире система управлениями версиями.

Что внутри?

Когда файл добавляется для отслеживания в Git, он сжимается с помощью алгоритма сжатия `zlib`. Результат хэшируется с помощью хэш-функции SHA-1.

Это даёт уникальный хэш, который соответствует конкретно содержимому в этом файле. Git хранит его в базе объектов, которая находится в скрытой папке `.git/objects`. Имя файла — это сгенерированный хэш, а файл содержит сжатый контент. Данные файлы создаются каждый раз при добавлении в репозиторий нового файла (или изменённой версии существующего файла).

Основные команды

- `git init`: инициализировать текущий каталог как репозиторий Git (создаётся скрытая папка `.git` и её содержимое).
- `git clone <git-url>`: загрузить копию репозитория Git по указанному URL.
- `git add <filename.ext>`: добавить неотслеженный или изменённый файл в промежуточную область (создаёт соответствующие записи в базе данных объектов).
- `git commit -m 'Commit message'`: зафиксировать набор изменённых файлов и папок вместе с сообщением о коммите.
- Если коммит ещё не был отправлен на сервер (push), то можно воспользоваться простой командой, позволяющей редактировать текст сообщения к последнему коммиту: `git commit –amend`
- `git status`: показать статус рабочего каталога, текущей ветви, неотслеженных файлов, изменённых файлов и т. д.
- `git branch <new-branch>`: создать новую ветвь на основе текущей извлечённой ветви.
- `git checkout <branch>`: извлечь указанную ветвь в рабочий каталог.
- `git merge <branch>`: объединить указанную ветвь с текущей, которая извлечена в рабочий каталог.
- `git pull`: обновить рабочую копию, объединив в неё зафиксированные изменения, которые существуют в удалённом репозитории, но не в рабочей копии.
- `git push`: упаковать свободные объекты для локальных коммитов активной ветви в файлы пакета и перенести в удалённый репозиторий.
- `git log`: показать историю коммитов и соответствующие сообщения для активной ветви.

Если что-то пошло не так

`git reflog -` вы увидите список всего, что сделали в git, во всех ветках, у каждого элемента есть индекс HEAD@{индекс} найдите тот, перед которым всё сломалось

`git reset HEAD@{index}`

либо

`git reset –hard HEAD~1 -` отменить последний коммит

отменить коммит, который был n коммитов назад

`git log`

`git revert [хэш того самого коммита]`

Useful links

https://initialcommit.com/cluster/git-commands-git-cheat-sheets

https://git-scm.com/