Системы контроля версий. Ветвление.

весна 2023

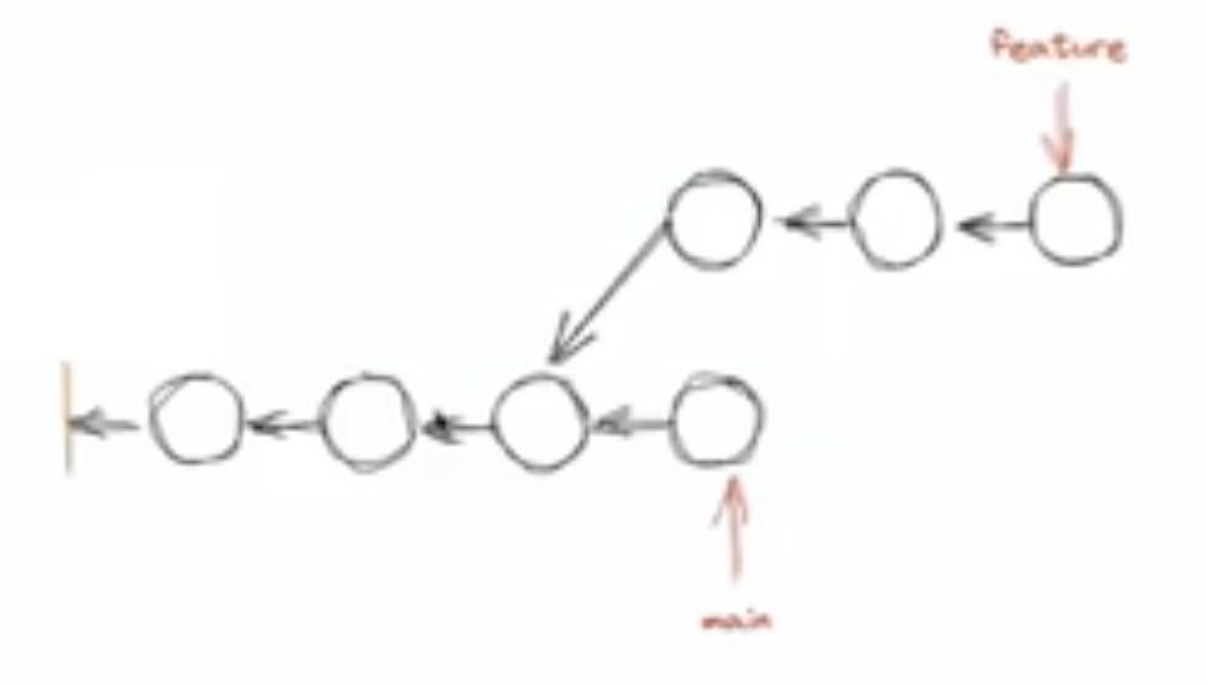
Branches

Когда мы работаем с проектом, часто нам нужно проводить какие-то изменения в проекте, проводить какие-то эксперименты, работать одновременно и параллельно.

Лучше этого не делать в той ветке, где у нас лежит основная версия кода. Код в основной ветке должен быть рабочим.

В таких ситуациях используются ветки.

Ветка в git - это указатель на какой-то commit



Посмотрим на git в консоли (продолжение)

> git branch “имя ветки”

> git log --oneline --all --graph

> vim hi.txt

> git checkout “имя ветки”

Когда мы переключаемся из ветки в ветку, если мы трэкаем все файлы, которые лежат на нашей файловой системе, git разворачивает соответствующее состояние проекта.

В процессе разработки регулярно возникают такие ситуации, когда основная ветка проекта, в которую происходят коммиты других разработчиков, уходит вперёд.

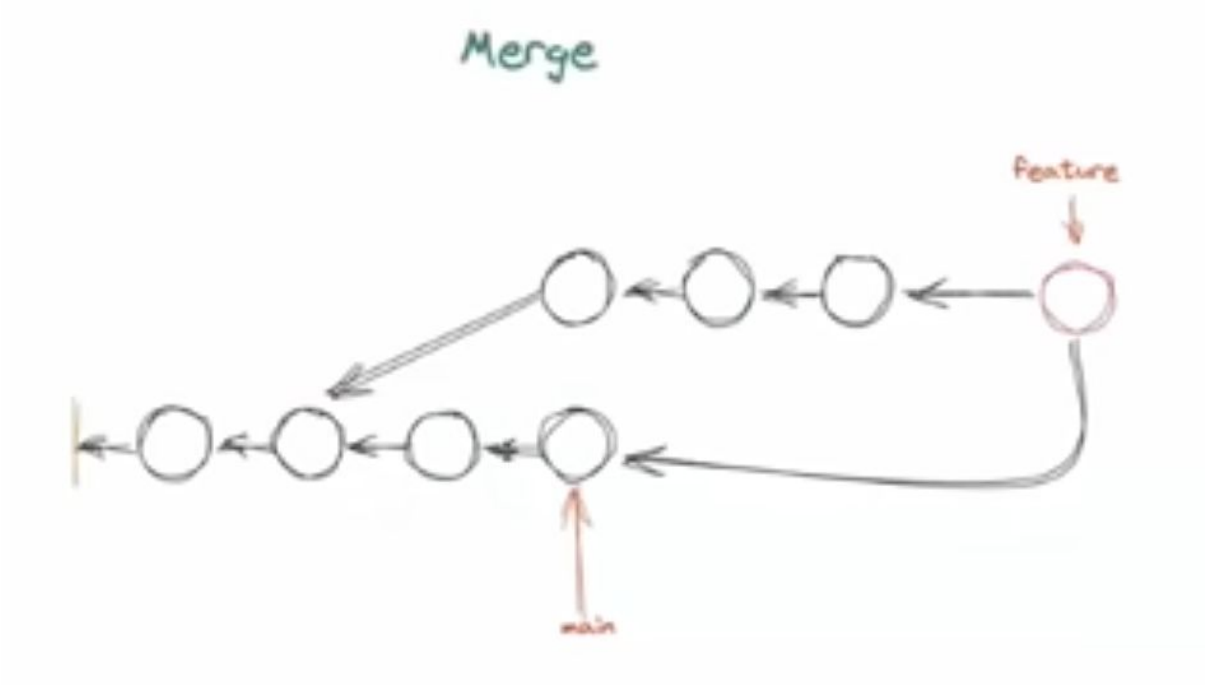
Естественно, ответвления не включают в себя те изменения, которое произошли после того, как были сделаны эти ответвления.

Merge

Слияние двух и более состояний и фиксация этого состояния в новом коммите.

После merge мы получим новое состояние, которое включает в себя то, что было в объединяемых ветках.

Merge



Посмотрим на git в консоли (продолжение)

> git checkout “имя ветки”

> git merge master

> git log --oneline --all --graph

Rebase

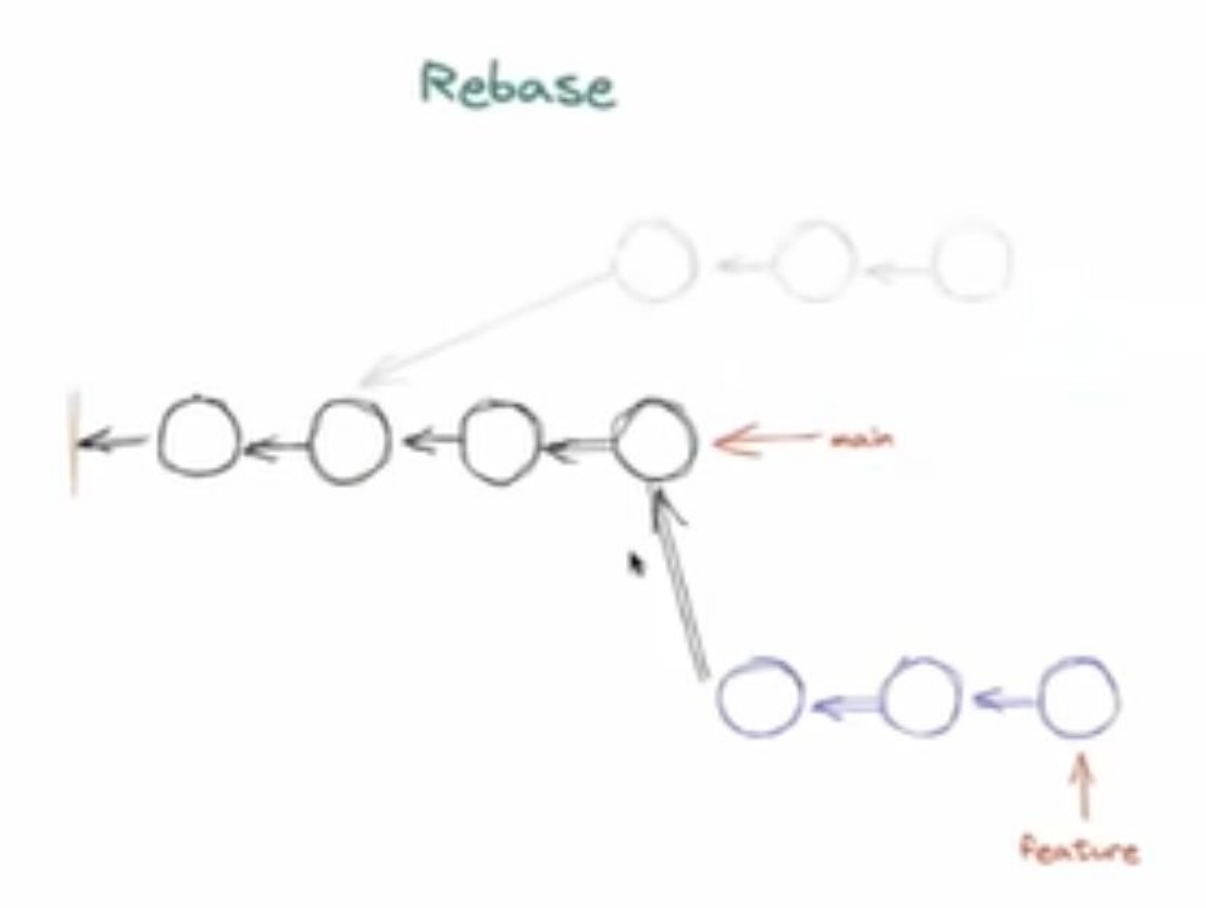
Копирует изменения, которые были в копируемой ветке, а сверху накатывает изменения.

Получается, что rebase переписывает историю конкретной ветки, базирует её на новом состоянии.

Можно использовать на локальной копии репозитория, но

в общем репозитории не надо делать rebase!

Rebase



Посмотрим на git в консоли (продолжение)

> git checkout “имя ветки”

> git log --oneline --all --graph

> git reset --hard

> git rebase master

Посмотрим на git в консоли (продолжение)

> git merge “имя ветки” --no-ff

no fast-forward

> git revert

> git merge “с конфликтами”

> git reset --hard HEAD^2~1

Remote = remote database

fetch, push, pull = synchronize local database

Remote - просто удалённая база данных, с которой мы синхронизируемся.

Удалённых баз (remote), с которыми происходит синхронизация может быть несколько.

Посмотрим на git в консоли (продолжение)

> cd ..

> mkdir remote-repo

> cd remote-repo

> git init --bare .

--bare означает, что не надо хранить рабочую копию, только объектную базу

Посмотрим на git в консоли (продолжение)

> git log --oneline --all --graph

> git remote add origin ../remote-repo/

origin - это может быть любое имя, но обычно используется

> git remote -v

Посмотрим на git в консоли (продолжение)

> git fetch

> git push origin master -u

флаг -u обозначает, что master трэкает origin

-u = --set-upstream

Посмотрим на git в консоли (продолжение)

```
> cd ..
> git clone remote-repo/ developer2
> cd developer2
> ls -la
> git log --oneline --all --graph
```

Посмотрим на git в консоли (продолжение)

> git fetch

> git push

> git pull --rebase=true

> git pull --rebase=false

github

https://docs.github.com/ru/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Правила совместной работы

Если у вас возникают проблемы синхронизации между командами, если у вас возникают конфликты в истории, скорее всего вы просто не выработали практики, позволяющие вам избегать этого.

Есть практики, которые помогают вам соблюдать гигиену, не напарываться на те проблемы, которые могут возникнуть, если у вас нету системы, если у вас нет договорённостей, и вы не знаете, что делать.

Git-flow и Trunk based development

https://www.atlassian.com/ru/git/tutorials/comparing-workflows/gitflow-workflow

https://www.atlassian.com/ru/continuous-delivery/continuous-integration/trunk-based-development

https://habr.com/ru/company/avito/blog/680522/