Оглавление

Английский язык

Английский язык

GIT

При разработке всегда используется удаленный сервер как резервная копия. Если вы работаете один, то у вас нет никаких проблем. Но при разработке проекта командой могут возникнуть проблемы. Что произойдет, если несколько человек внесут какие-то изменения и загрузят их на удаленный сервер? Скорей всего на удаленном сервере останутся изменения только одного человека.

Зачем нужен GIT:

- Берет на себя слияние разных версий файлов.
  Процесс называется merging. Может проводиться автоматически или в ручном режиме.
- Является системой контроля версий.
  У нас есть вся история изменений файлов. Мы можем вернуться к любой версии нашего проекта.

В системе контроля версий существует два подхода к хранению данных: централизированный и распределенный. При централизированном подходе проект храниться только на центральном сервере. При распределенном подходе проект храниться на центральном сервере плюс

у каждого разработчика есть копия проекта. Второй подход имеет преимущества, т. к. разрабатывать можно офлайн и, если что-то случиться с центральным сервером, то в этом случае копия проекта останется у разработчиков. Git является распределенной системой.

GIT — распределенная система контроля версий.

В отличии от других систем контроль версий, которые хранят список изменений в файле, GIT хранит изменения снимков проекта во времени.

Статусы файлов:

- untracked (неотслеживаемый) — файл создан;
- modified (измененный) — файл изменен;
- staged (подготовленный) — `git add .`;
- committed (зафиксированный) — `git commit`.

Указатели

В GIT есть указатель HEAD. Обычно он указывает на последний (текущий) коммит. Этот указатель можно смещать: `HEAD^` или `HEAD~1` (1 коммит), `HEAD^^` или `HEAD~2` (2 коммита) и т. д. Возвращает проект можно и по указанному хэшу коммита.

Работа с удаленным репозиторием

С помощью GIT можно отправить нашу работу на удаленный репозиторий — для дополнительной сохранности и для того, чтобы другие люди могли видеть наши коммиты.

Удаленные репозитории: GitHub, BitBucket, GitLab. Они предоставляют нам всю инфраструктуру для хранения и управления GIT-репозиториев.

Также может существовать несколько удаленных репозиториев. При помощи команды `git remote add` можно добавить несколько удаленных репозиториев (ссылки): дать им разные имена, и они будут находиться по разным адресам.

SSH

SSH (от анг. «Secure Shell» — «безопасная оболочка») — сетевой протокол, позволяющий производить удаленное управление операционной системой. SSH позволяет безопасно передавать данные в незащищенной среде.

В простом представлении работу SSH можно представить наличием приватных ключей у клиента (локального компьютера) и сервера.

Ссылки для настройки SSH-ключа для github.com:

- проверка наличия SSH-ключа;
- генерация SSH-ключа;
- связка SSH-ключа.

Ветвление

Зачем использовать ветвление:

- новые функции разрабатываются в отдельных ветках;
- ветка master содержит стабильную версию проекта, можем вернуться на master в любой момент;
- сразу несколько разработчиков могут работать в своих ветках над своими задачами, после завершения работы над задачами эти ветки «сливаются» в ветку master.

Fast-Forward merge:

- пока мы работали в своей ветке, в ветке master ничего не произошло (не было новых коммитов);
- GIT очень легко слить ветку add-feature1 в master (не может возникнуть конфликтов);
- не создается отдельный commit для слияния (merge commit).

~~Fast-Forward merge~~:

- пока вы работали в своей ветке, кто-то добавил коммиты в ветку master;
- или вы сами добавили новые коммиты в ветку master (пример — вас попросили исправить какой-нибудь критический баг и запушить на GitHub);
- могут возникнуть конфликты, гит попробует решить их самостоятельно, если у него не получится, придется решать их вручную;
- merge commit создается.

Если мы изменили один и тот же файл, то в этом случае происходит конфликт слияния. В этом случае GIT не может самостоятельно слить ветки. Необходимо решить конфликт вручную.

rebase

rebase — альтернатива merge:

- обе команды делают одно и то же — сливают ветки;
- команда merge может создавать merge commit при слиянии (в случае не fast-forward), команда rebase merge commit'а не создает;

- команда merge безопасней, чем rebase — есть отдельный commit, отображающий слияние;
- плюс merge — достоверная полная история commit'ов;
- плюс rebase — лаконичная линейная история без лишних коммитов;
- если в ветке долго велась работа и произошло много изменений лучше использовать merge;
- если ветка была недолгая и произошло мало изменений — можно использовать rebase;
- используйте merge, если вас не просят о rebase.

Команда rebase работает так, будто мы только сделали git pull и сразу добавили в нее изменения. Можно сказать, что новая ветка "перебазировалась" на последний коммит. Или в новую ветку был добавлен последний коммит из master, а затем, поверх него были добавлены коммиты текущей ветки. Теперь можно делать fast-forward слияние без merge commit'a.

После совершения данной команды коммиты текущей ветки помещаются во временную зону, далее в текущую ветку добавляются все коммиты из ветки master, позже

поочередно добавляются все коммиты из временной зоны.

Также можно сделать все наоборот. Можно перейти в ветку мастер и совершить текущую команду из нее. Таким образом сначала добавятся коммиты из новой ветки, а затем новый коммит из мастера.

Разрешение конфликта такое же, как в случае с merge.

Интерактивный rebase

- обычный rebase нужен для манипуляций с ветками, интерактивный rebase работает на одной ветке;
- обычный rebase берет коммиты из другой ветки, перемещает их в нашу ветку и поверх этих коммитов по одному применяет коммиты из временной зоны;
- интерактивный rebase не берет коммиты из другой ветки, он помещает некоторые коммиты из текущей ветки во временную зону и потом применяет эти коммиты опять к текущей ветке (в момент применения мы можем изменить коммиты);
- несмотря на то, что название команд одинаковое, обычный rebase сильно отличается от интерактивного rebase (разная логика).

Интерактивный rebase работает с коммитами, которые идут после того коммита, который вы указали.

Что можно делать с помощью интерактивного rebase:

- поменять коммиты местами;
- поменять название коммита(ов);
- объединить два коммита в один;
- добавить изменения в существующий коммит;
- разделить коммит на несколько коммитов;
- ...

Команды

Информационные команды:

git help (помощь, документация);

git help название_команды (документация конкретной команды).

Конфигурация:

git config --global user.name "имя фамилия";

git config --global user.email "email";

git config --global color.ui true.

Создание нового проекта:

mkdir название_проекта (создание каталога);

cd название_проекта (перейти к данному каталогу);

git init (инициализация репозитория git).

Базовые команды:

git status (узнать текущий статус репозитория);

git add (подготовить файлы к коммиту);

git commit (сделать коммит).

git log (история коммитов);

Другие команды:

git diff (разница между текущим неотслеживаемым состоянием репозитория и последним снимком репозитория);

git reset (отмена изменений, откату к снимку);

git clean (удаление untracked файлов);

git checkout (перемещения между коммитами, версиями отдельных файлов и ветками);

git remote (настройка и просмотр удаленных репозиториев);

git push (отправка локального репозитория на удаленный);

git pull (git fetch, git merge) (получения обновлений (новые коммиты) с удаленного репозитория);

git clone URL_репозитория (загрузить репозиторий);

git branch (работа с ветками);

git merge ветка (слияние текущую и указанную веток);

git rebase ветка (слияние текущую и указанную веток, разница команд описана выше);

git rebase --continue (принять команду после исправления вручную конфликтов);

git rebase --skip (пропустить коммит, который вызывает конфликт слияния);

git rebase --abort (прекратить слияние);

git rebase -i HEAD~3 (интерактивный rebase);

git cherry-pick ("взять" коммиты из другой ветки).

Источник

Курс на Udemy «Git: Полный курс для начинающих и не только», Наиль Алишев, 03.2019.

g

Операционная система

Linux (bash)

Windows (bat)

Паттерны и алгоритмы

Паттерны и алгоритмы

Java Core

Java Core

Инструменты

Инструменты

Фреймворки

Web basic

Web basic

Тестирование

Тестирование

Utils

Utils

База данных

База данных

Java Performance

Java Performance