GIT

Чувакин Сергей

R meetup

4 февраля 2021 г.

Outline

Что это и зачем?

- Формально - ГИТ это система контроля версий.
- Неформально - это программа, которая позволяет содержать код в «чистоте». Особенно полезно, когда у вас большой проект. Это наверное единственный способ максимально эффективно работать с кодом в распределенной команде.

Что это и зачем?

Почему?

GIT был разработан Линусом Торвальдсом, и является де факто стандартом в разработке. Он вам нужен, если:

- Ваша кодовая база достаточно большая, чтобы держать все в голове
- Ваш код важен и вы за него переживаете
- Вы работаете в команде
- Вы хотите делиться кодом с другими людьми
- У вас несколько версий проекта
- Вы работаете с продакшеном - но в этом случае, вы скорее всего уже все знаете

Начальный Глоссарий

- Сервер - буквально это компьютер, у него есть мощности, есть память, есть ОС
- Локальный сервер - ваш компьютер
- Удаленный (remote) сервер - компьютер где то далеко. Вы к нему подключаетесь с помщью специальных тилит и протоколов передачи данных
- git - технология (с которой мы знакомимся)
- github - сервих для работы с гитом

Сценарий 1

Версионирование локального кода

```
git init # начало
git log # история
git status # статус!
```

Сценарий 1

Версионирование локального кода

Обращайте внимание на подсказки - они полезные. Следующим действие гит предлагает сделать add.

- add - это команда, которая отправляет файл в staging
- staging - это состояние отслеживания файла.

Сценарий 1

Версионирование локального кода

После staging изменения в файле можно записать, запомнить или иначе говоря *закоммитить*!

- коммит (англ. commit) - это кирпичик вашей истории изменений файла
- если вы не работаете с github, комитами можно ограничится
- все коммиты имеют уникальный номер - хэш (англ. hash)
- именно по хэшу можно откатиться назад

Сценарий 1

Версионирование локального кода

NB: данные нельзя коммитить! Создайте файл .gitignore и запишите туда путь в вашим данным.

```
touch .gitignore
echo my/data/path >> .gitognore
```

Сценарий 1

Версионирование локального кода

Следующие шаги:

```
git add my_script.R # или просто git add .
git commit -m 'initial commit' ## Сообщение обязательно!
```

Сценарий 1

Версионирование локального кода

Проверьте снова:

```
git log
git status
git diff <commit hash> # здесь можно посмотреть
# изменения внесенные коммитом
```

Сценарий 1

Версионирование локального кода

Откатимся назад!

```
git reset <commit hash>
```

Важно понимать, что мы откатываемся *к состоянию (HEAD) на момент коммита*.

Пример:

```
git reset 5a0c49c17e1b83227ba4a3acf2a179592c6f378c
```

Сценарий 1

Версионирование локального кода

Откатиться можно по разному:

- --soft - настоящее состояние записывается в stage
- --mixed (default) - настоящее состояние не записывается в stage
- --hard - настоящее состояние удаляется

Сценарий 2

Синхронихзация с удаленным сервером

- Перейдем на github и создадим репозиторий
- Теоретически мы можем связать два репозитория, но это не самый простой способ
- Куда лучше склонировать уже существующий репозиторий , там настроены все связи
- Удаленный репозиторий по сути то же самое, что и папка, которую мы создавали для проекта

Сценарий 2

Синхронихзация с удаленным сервером

В целом сценарий здесь тот же, мы работаем с кодом, а затем:

```
git add .
git commit -m 'message'
git push # на сервер!
```

В остальном, все то же самое, за исключеним того, что нужно в конце добавлять команду

```
git push
```

Сценарий 2

Синхронихзация с удаленным сервером

Если вы что то изменили прямо на удаленном сервера, то «стянуть» изменения на локальный компьютер можно через:

```
git pull
```

Сценарий 2

Синхронихзация с удаленным сервером

Клонирование репозитория на локальный компьютер. После клонирования - на компьютере появляется папка со всеми внутренними настройками и историей коммитов.

```
git clone https://github.com/bradtip/Awesome_project.git
```

*Клонирование - скачивание удаленного репозитория себе на компьютер.

Сценарий 2

Синхронихзация с удаленным сервером

Recap: В целом, когда вы работаете с удаленным сервером - все то же самое, только добавляются две новые команды:

```
git push
git pull
```

Сценарий 3

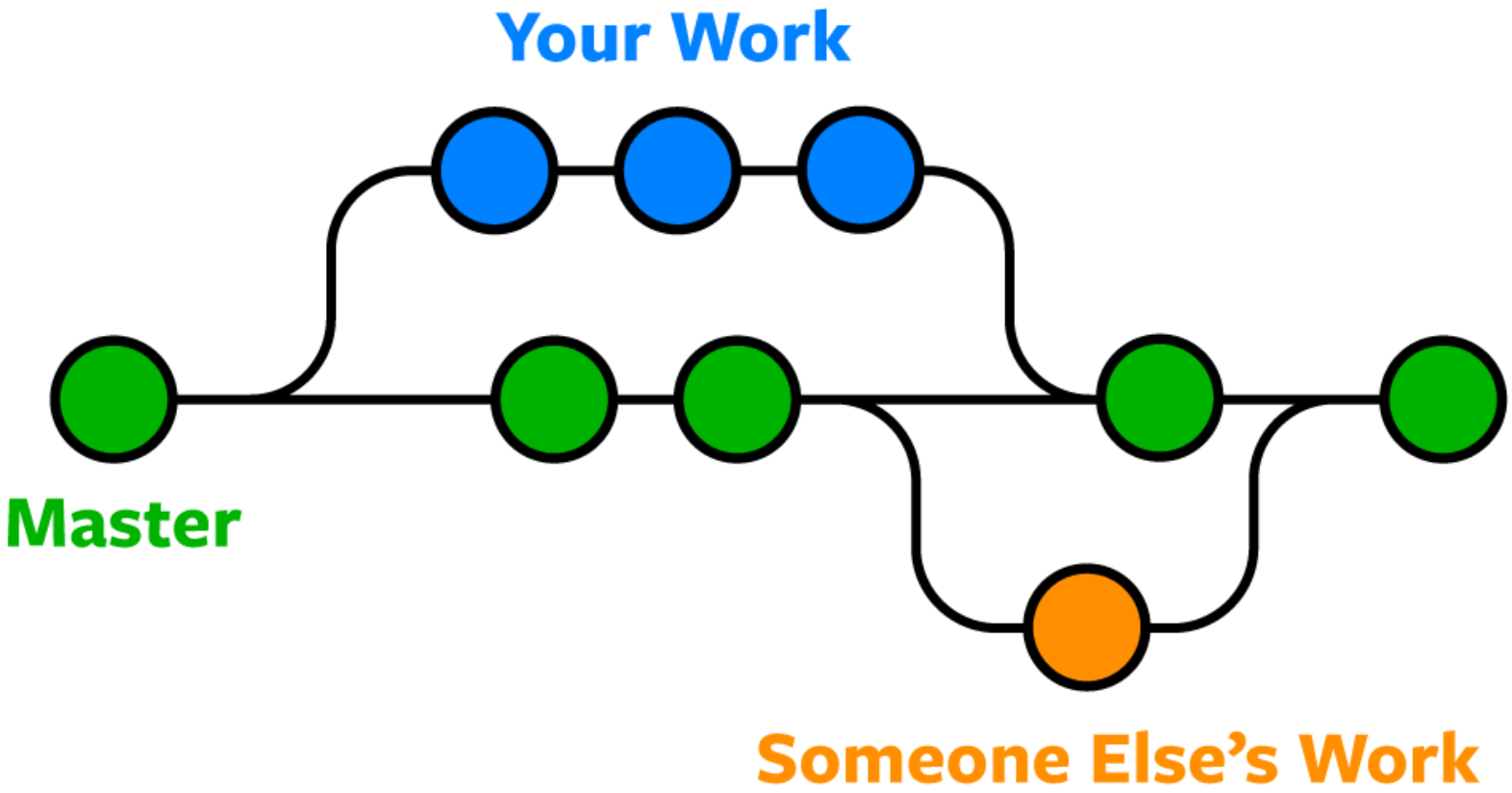
Работа в команде

Главное - в команде можно и нужно работать с помощью ГИТ.

Бранчи (с англ. branch) - ответвления развития кода. Мы помним - что попрежнему отдельный кирпичик - это коммит.

Сценарий 3

Работа в команде



Сценарий 3

Работа в команде

По преженему скажу, что лучше всего создавать ветки через github, но можно и через командную строку.

Ветка - это полноценная копия вашего репозитория, но со своими изменениями. При клонировании ропозитория, ветки подтягиваются автоматически.

Сценарий 3

Работа в команде

Команды с ветками:

```
git branch # список
git branch <name> # создание
git branch -d <name> # удаление
git checkout <name> # переключится
git push -u origin <branch> # послать на сервер
git fetch # стянуть новые ветки с сервера
git merge <branch> # соеденить ветки
```

NB: могут быть конфликты - их нужно будет решать! от этого никуда не деться!

Сценарий 3

Работа в команде

Пул-реквест - это спецаильная процедура слияния веток. Именно процедура, которая доступна только через github или другой графический аналог (bitbucket, gitlab). Это нужно скорее для того, чтобы соответсвовать процессам разработки.

Best Practices

Несколько советов и правил:

- Нельза коммитить данные (и большие файлы), для этого есть другие технологии, это очень важно! (e.g. dvc)
- При создании веток используйте графический интерфейс, меньше вероятность допустить ошибку
- При создании репозитория используйте графический интерфейс, меньше вероятность допустить ошибку
- Познакомьтесь с функционалом vs code, там много инструментов, который помогают эффективно работать с гитом (git lens, git history, git blame)
- Не пишите пустых и бестолковых сообщений, описывайте так, чтобы было понятно, что в коммите