**Для чего нужен Git.**

**Система контроля версий - для чего она нужна вашему проекту**

Это система, регистрирующая изменения документов на сайте, благодаря которой можно вести удобную совместную разработку, также она дает возможность вернуться к предыдущим или альтернативным версиям сайта. Повышает безопасность, поскольку нет необходимости давать доступ к рабочему сайту, а можно использовать для всех доработок СКВ (систему контроля версий).

**Зачем нужна система контроля версиями**

- можно легко отследить ошибки.

- изменения сохраняются (когда, кем и что было сделано).

- исключена поломка сайта.

- возможность легко откатывать изменения до любой версии сайта.

- возможность использовать как простой инструмент для переноса данных с сайта разработки на рабочий сайт.

**Как происходят изменения в проекте без контроля версий**

Исходя из практики, мы видим, что работа с сайтом в большинстве случаев проходит таким образом: при необходимости внести изменения в проект, они вносятся сразу на самом проекте, так как это быстрее. Кто-то вносит их на тестовой версии, но тут мы сталкиваемся, со сложностью по актуализации файлов. При возникновении ошибки — быстренько разворачиваем  бекап, но тогда все изменения не сохранятся, и придется все опять вносить.

Конечно, использование тестовой версии более безопасно, но в таком случае, много ручной работы и не стоит исключать человеческий фактор.

* Если изменений много, программисту придется собрать список всех измененных файлов.
* Программист работает в паре с другим программистом или верстальщиком и правят один файл. Как результат, кто последний сохранил тот и молодец.
* Разработчик проделал огромную работу по сайту. Теперь нужно просто заменить все файлы на боевом сайте на новые. Но внезапно ему сообщают, что на сайте только что закончили работу по настройке ЧПУ, и, скорее всего, код разработчика работать не будет.
* Даже если ошибка всплывет только в понедельник, программист без проблем вспомнит, в какой строчке кода сделал ошибку, их там всего-то несколько сотен тысяч.

В данном случае, самым безопасным и простым способом решения будет — использование системы контроля версий ее необходимо установить один раз. Но в дальнейшем это избавит от ряда неудобств.

**Схема работы системы контроля версий**

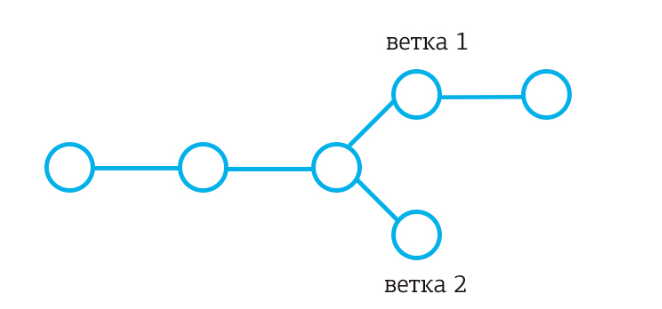
Любое завершенное действия программиста и верстальщика фиксируется системой. Как результат, есть история изменения всех файлов. И при необходимости можно вернуться на любой файл.

Для этого используем git, который устанавливаем на сервере. Тут нам понадобится SSH-доступ и удаленный репозиторий.

Мы используем бесплатную альтернативу – GitLab.

**Как работать с GIT**

Необходимо создать для каждой задачи соответствующую ветку. Благодаря чему, каждый программист работает с сайтом в своей ветке, при этом, не мешая, работе друг друга.



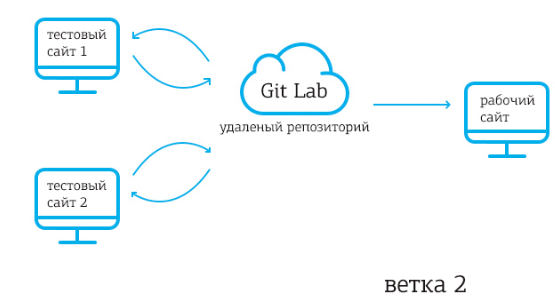
Любое логическое завершение задачи фиксируется с комментарием. Для переноса изменений необходимо отправить все на удаленный репозиторий.

После чего, вебхук видит команду и пускает в работу скрипт на основном сайте и автоматически обновит файлы.

В работе нескольких человек перед отправкой на рабочий сайт, необходимо разрешение с ведущим программистом.

Поэтому доступ к основной ветке следует запретить. А соединение с ней будет происходить по средству запросов (merge request).

После проверки ведущим программистом, если не требуется доработка, и нет правок, система принимает код, который с помощью вебхука отправляется на рабочий сайт.



После слияния ветки срабатывает вебхук, запускающий на рабочем сайте код, который в свою очередь автоматически обновляет содержание сайта, выполняя две простые команды:

1. Забирает все изменения из удаленного репозитория

git fetch --all

1. Обновляет файлы, до состояния зафиксированном на удаленном репозитории, при этом удаляя все изменения на сайте, которые не были зафиксированы.

git reset --hard origin/master

Конфликты возникают, когда произошли изменения в одном и том же файле, и при слиянии (merge) git не может решить, как правильно объединить файлы. Если одновременно редактировалась одна строка, то возникает конфликт. Программно невозможно решить, какая из версий должна быть на сайте.

**Как пользоваться Git.**

Что бы пользоваться git необходимо зарегистрироваться, скачать с официального сайта программу. Установить, и с помощью Git Bash пользуясь командами создавать и работать с локальным и глобальным репозиториями Git.

**Основные команды Git.**

|  |
| --- |
| ==0 урок== |
| git --version |
| git config --global user.name "\*\*\*" |
| git config --global user.email "\*\*\*" |
| cd /\*\*\*/ |
| mkdir git-sample |
| cd git-sample |
| git init |
| ls - a |
| cd .git/ |
| clear |
| cd .. |
| git config user.name "Test user" |
| git config user.email "Test@mail.ru" |
|  |
| ==1 урок== |
| nano test.txt |
| -> Hello Git |
| git add test.txt |
| gitk |
| git commit |
| git commit -m "First coommit" |
| gitk |
| -> 123 line |
| git commit |
| git add test.txt |
| git commit -m "Secondary coommit" |
| gitk |
|  |
| ==2 урок== |
| git status |
| ->add new line |
| clear |
| git status |
| git add test.txt |
| git status |
| git commit -m "test status" |
| git status |
|  |
| ==3 урок== |
| ->index.txt <h1> Hello Git</h1> |
| git status |
| git add index.txt |
| git status |
| git commit -m "New file" |
| ->123123 .html |
| git status |
| git add \* |
| git status |
| git add index.txt |
| git status |
| git commit -m "New commit" |
| gitk |
|  |
| ==4 урок== |
| git log |
| git log --pretty=oneline |
| git log --pretty=oneline --max-count=3 |
| git log --pretty=oneline --all |
| git log --pretty=oneline --author="Test user" |
| git log --pretty=format:"%h - %s : %ad [ %an ]" |
| git log --pretty=format:"%h - %s : %ad [ %an ]" --date=short |
|  |
| ==5 урок== |
| git log --pretty=format:'%h %ad | %s%d' --date=short |
| git checkout \*\*\* ///new file |
| ls |
| gitk |
| git checkout ///test status |
| ls |
| git checkout master |
| gitk |
|  |
| ==6 урок==Отмена индексированных файлов |
| ls |
| nano test.txt |
| -> fdsafsdf |
| git status |
| git add test.txt |
| git status |
| ls |
| nano index.html |
| ->eqfdasdfads |
| git status |
| git add index.html |
| git status |
| git reset HEAD test.txt |
| git status |
| git checkout test.txt |
| git status |
|  |
| ==7 урок==Revert - Отмена коммита |
| ls |
| nano index.html |
| ->asdfasdas |
| git status |
| git add . |
| git status |
| git commit -m "Wrong commit" |
| git status |
| nano index.html |
| git log |
| git revert HEAD |
| git revert HEAD --no-edit |
| git status |
| git log |
| nano index.html |
| git revert \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --no-edit ////new file |
| git status |
|  |
| ==8 урок==Решение простого конфликта |
| git status |
| ////git revert --abort |
| nano index.html |
| git add index.html |
| git revert --continue |
| nano index.html |
| ->11212afsdf |
| git add index.html |
| git commit |
| git status |
| git log |
|  |
| ===Урок 9. Ветки и их применение=== |
| git branch |
| git checkout -b newbranch |
| git branch |
| git checkout -b dev |
| git checkout -b f1 |
| git checkout -b f2 |
| git checkout -b f3 |
| git branch |
| git checkout dev |
| ls |
| nano test.txt |
| git status |
| git add test.txt |
| git status |
| git commit -m "Test branch" |
| git log |
| git checkout master |
| git log |
|  |
| ==Урок 10. Слияние веток и решение конфликтов слияния== |
|  |
| git checkout dev |
| nano devfile.txt ///type or edit |
| -> Hello Devcolibri |
| ls |
| git status |
| git add devfile.txt |
| git commit -m "Added new file" |
| ls |
| git checkout master |
| ls |
| git merge dev |
| git log |
| git status |
| nano devfile.txt |
| -> my change |
| git status |
| git add devfile.txt |
| git commit -m "Change file" |
| git log |
| git checkout dev |
| nano devfile.txt |
| ->hello first change |
| git checkout master |
| git add devfile.txt |
| git commit -m "Add new line" |
| git checkout master |
| git status |
| git commit -m "Add new line to file" |
| git status |
| git checkout master |
| git merge dev |
| git status |
| nano devfile.txt |
| -> redachim |
| git status |
| git add devfile.txt |
| git commit |
| :wq |
| git status |
| git log |
|  |
| ==Урок 11. Rebase vs. Merge - Что такое git rebase?== |
| git branch -a |
| git log |
| git checkout dev |
| ls |
| nano test.txt |
| ->dasdadaqwwq |
| ls |
| git status |
| git add . |
| git commit -m "Add some change" |
| git checkout master |
| git rebase dev |
| git log |
| ls |
| nano index.html |
| -> <p>test</p> |
| git add . |
| git commit -m "Commit -1" |
| git checkout dev |
| ls |
| nano test.txt |
| ->112121 |
| git status |
| git add . |
| git commit -m "Commit -2" |
| git checkout master |
| git rebase dev |
| git log |
| JOPA |
|  |
| ==Урок 12. Удаленный репозиторий и git clone== |
| git clone ->URL |
| cd |
| git status |
| git log |
|  |
|  |
| cd |
| ls |
| git remote add origin https://github.com/Slavik6661/git-sample.git |
| git push -u origin master |
|  |
| ==Урок 13. Загружаем изменения на удаленный репозиторий== |
| git status |
| git push |
| nano |
| git add . |
| git status |
| git push |
| git commit -m "Removed on line" |
| git branch |
| git checkout newbranch |
| git log |
| git push origin newbranch |
|  |
|  |
| ==Урок 14. Обновляем код манерой merge и rebase== |
| git pull origin master |
| git pull --rebase origin master |
|  |
| ==Git: Урок 15. Что такое Origin и для чего нужен stash?== |
| ls |
| nano |
| git status |
| git pull --rebase origin master |
| git pull origin master |
| git pull --rebase origin master |
| git stash |
| git status |
| git pull --rebase origin master |
| nano |
| git stash pop |
| git stash list |
| git stash show |
| git stash drop |
| git stash apply |

**Вывод**

Если с вашим сайтом работает несколько программистов, для исключения ошибок необходимо использовать систему контроля версий. В таком случае вы обезопасите себя от ошибок, которые могут возникнуть в процессе работы.

**SmarGit**

