**Лабораторная работа №19-20**

**Тема:** «Изучение систем контроля версий GIT, GITHUB, GitLAB и др».

Системы контроля версий git:  
  
1.Опишите назначение систем CVS.

**CVS** — централизованная [система управления версиями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8), популярная в 1990-е — начале 2000-х годов. Хранит историю изменений определённого набора файлов, как правило, [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), и облегчает совместную работу группы людей над одним проектом. Распространяется на условиях [лицензии](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D1%8F) [GNU GPL](https://ru.wikipedia.org/wiki/GNU_General_Public_License).

2. Какие виды CVS вам известны? Какими особенностями они обладают?

### **Система одновременных версий (CVS)**

Изначально CVS была создана, чтобы избежать конфликта версий. Всем участникам для работы предоставлялась только самая последняя версия кода. Это была первая система контроля версий. Пользователю нужно было быстро фиксировать изменения в репозитории, пока другие не опередили его. Данная технология испытана временем и удерживается на рынке десятки лет.

### **Apache Subversion (SVN)**

Как и CVS, SVN это бесплатная система контроля версий с открытым исходным кодом. С той лишь разницей, что распространяется под лицензией Apache, а не под Открытым лицензионным соглашением GNU. Для сохранения целостности базы данных SVN использует так называемые атомарные операции. При появлении новой версии к финальному продукту применяются либо все исправления, либо ни одно из них. Таким образом, код защищают от хаотичных частичных правок, которые не согласуются между собой и вызывают ошибки. Из особенностей:

* Система на основе CVS
* Допускает атомарные операции
* Операции с ветвлением кода менее затратны
* Широкий выбор плагинов IDE
* Не использует пиринговую модель

### **Git**

Эта система была создана для управления разработкой ядра Linux и использует подход, который в корне отличается от CVS и SVN.

### **Mercurial**

[Mercurial](http://mercurial.selenic.com/) была выпущена одновременно с Git. Это  также распределенная система контроля версий. Mercurial создавалась в качестве альтернативы Git для разработки модулей ядра Linux. Но так как выбрали все-таки Git, то Mercurial используется меньше. Тем не менее, многие ведущие разработчики работают именно с этой системой, например [OpenOffice.org](http://www.openoffice.org/). Система контроля версий Mercurial отличается от других систем контроля версий тем, что главным образом она написана на Python (а не С). Однако, некоторые части выполнены в качестве модулей-расширений на C. Из особенностей:

* По сравнению с Git легче в освоении
* Подробная документация
* Распределенная модель системы контроля версий

3. На примере системы контроля версиями GIT опишите особенности подхода к работе с данными. Какими преимуществами данная система обладает?

В основу [Git](http://git-scm.com/) закладывались концепции, призванные создать более быструю распределенную систему контроля версий, в противовес правилам и решениям, использованным в CVS. Так как Git разрабатывалась главным образом под Linux, то именно в этой ОС она работает быстрее всего. Git также работает на Unix-подобных системах (как MacOS), а для работы на платформе Windows используется пакет mSysGit.

**Преимущества:**

* Прекрасно подходит для тех, кто ненавидит  CVS/SVN
* Значительное увеличение быстродействия
* Дешевые операции с ветками кода
* Полная история разработки доступная оффлайн
* Распределенная, пиринговая модель

4. Опишите процесс установки И настройки GIT на свой ПК .

1) Переходим на официальный сайт Git  
2) Выбираем нужную версию для вашей ОС.  
3) Запускаем установочный файл и устанавливаем Git.  
4) Открываем терминал (Linux и MacOS) или консоль (Windows) и вводим следующие команды:  
git config —global [user.name](http://user.name/) "<ваше\_имя>" - установка имени  
git config —global [user.email](http://user.email/) "<[адрес\_почты@email.com](http://%D0%B0%D0%B4%D1%80%D0%B5%D1%81_%D0%BF%D0%BE%D1%87%D1%82%D1%8B@email.com/)>" - установка e-mail

GITHUB

1. Опишите назначение сервиса GitHub.

**GitHub** — это хостинг: он позволяет хранить проекты удалённо на сервере и работать с ними из любой точки мира. Доступ к файлам есть у всех, у кого есть ссылка. Одна из главных функций **GitHub** — контроль версий. Все изменения в коде можно отследить, поэтому в командной разработке это незаменимая вещь.

1. Опишите назначение сервиса Github Desktop.

**GitHub Desktop** — программа для удобной работы с проектами, размещенными на одноименном сервисе. С ее помощью можно добавить любые репозитории сразу на жесткий диск, обычным перетаскиванием или через проводник. Если перетаскиваемая папка является **Git** репозиторием, то **GitHub** импортирует всю историю и подсоединит её к профилю пользователя.

1. Поясните термины git и github:

**Git** — это программное обеспечение для отслеживания изменений в любом наборе файлов, обычно используемое для координации работы программистов, совместно разрабатывающих исходный код во время разработки программного обеспечения.  
**GitHub** — это служба размещения в Интернете репозиториев Git.

1. Репозиторий

**Репозито́рий** — место, где хранятся и поддерживаются какие-либо данные. Чаще всего данные в репозитории хранятся в виде файлов, доступных для дальнейшего распространения по сети.

1. Рабочая область и хранилище

**Корневая папка проекта** — это рабочая область. В ней находятся все файлы и папки, необходимые для его работы.

**Хранилище** — это содержимое скрытой папки. git. В этой папке хранятся все версии рабочей области и служебная информация.

1. Форк

**Форк** — полная копия мастер-репозитория, в которой вы можете безопасно работать.

1. Ветка

В **Git ветки** — это элемент повседневного процесса разработки. По сути ветки в Git представляют собой указатель на снимок изменений. Если нужно добавить новую возможность или исправить ошибку (незначительную или серьезную), вы создаете новую ветку, в которой будут размещаться эти изменения.

1. Мастер

**Мастер** — ветка по умолчанию. Когда идёт создание коммитов на начальном этапе дана ветка master, указывающая на последний сделанный коммит. При каждом новом коммите она сдвигается вперёд автоматически.

1. Коммит

**Коммит** — это снимок локальных файлов, записанный в локальный репозиторий.

1. Пул

**Git pull** используется для извлечения и загрузки содержимого из удаленного репозитория и немедленного обновления локального репозитория этим содержимым.

1. Пуш

**Git push** используется для выгрузки содержимого локального репозитория в удаленный репозиторий.

1. Пулрекквест

**Пул-реквест** — это запрос (англ. request — «запрос») на интеграцию изменений из одной ветки в другую.

1. Мердж

**Git merge** выполняет слияние отдельных направлений разработки, созданных с помощью команды git branch, в единую ветку.

1. Кодревью

**Code Review** — это процесс проверки и анализа кода задачи разработчиком перед ее релизом.