**Agile –** методологія гнучкої розробки програмного забезпечення. Робить акцент на контролі якості рзробки чи підтримки ПО. Стадії: аналіз, проектування, програмування, документування, тестування. Використовуються так-звані спрінти з їхніми ітераціями. З беклогапроекта вибираються задачі, які на себе бере команда і виконує спринт. Створюється бек-логспрінта. Задача оцінюється в людино-годинах. Вирішення однієї задачі триває не більше одного дня. Кожного дня відбуваються звіти – що зроблено і що буде зроблено. Задіяні «свині» і «купи», на манер жарта про ретсоран.

**Хешування** – перетворення за певним алгоритмом вхідного потоку даних довільної довжини в вихідну бітову строку фіксованої довжини. Такі перетворення також називають **х еш-функціями**, а їх результати **хешом**, **хеш-кодом**, **хеш-сумою**. Використовуються в криптографії, хеш-таблицях, тощо.

**Контрольна сума** – хеш-фунція, яка вираховує контрольний код який застосований для попередження помилок зберігання та передачі даних. Мають високу швидкість в порівнянні з криптографічними хеш-функціями але не є криптостійкими.

**Асоціативний масив** – абстрактний тип даних, який зберігає в собі пару ключ-значення. Шукає пари по ключу.

**Хеш таблиця** – структура даних, яка реалізує асоціативний масив зберігаючи пару ключ-значення. Має функція добавлення пари, пошуку та видалення пари по ключу. Види: з ланцюгом (масив з списком пар) та відкритою адресацією (масив з самим парами).

**Колізія** – ситуація, коли для різних ключів отримується одинаковий хеш.

**Тег** (дескриптор) – елемент розмітки гіпертексту. Є відкриваючий і закриваючий теги.

**Хештег** (#, шарп) – в різних мовах використовується за різним призначенням. В CSSвибір елемента по ID. В HTMLкод кольору. В С, С++ - для в казання на дериктиви процесора компілятора. В просторах інтернету використовують для об’єднання певної інформації в загальну групу по хештегу.

**TDD (testdrivendevelopment)** – методологія розробки при якій спочатку пишуться тести до модула, а потім розробляється сам модуль.

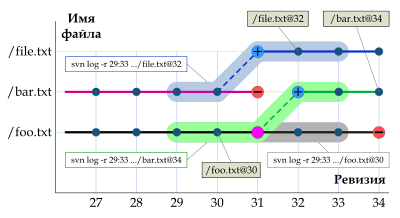
**Subversion (SVN)** – централізована система керування версіями. (на відміну від розгалуженої Git). Клієнти копіюють об’єкти з сервера створюючи їх локальні копії, роблять зміни і фіксують їх на сервері. Для доступу декількох клієнтів до сховища одночасно використовується модель *копіювання-зміна-злиття.*А для файлів які не допускають злиття (таких як бінарні файли) використовується модель *блокування-зміна-розблокування*. При збереженні змін відбувається дельта компресія (фіксується відмінність нової версії від старої і зберігаються тільки зміни, а не копіюється весь код заново).

Є 2 типа репозиторіївSVN – для збереження баз даних (на основі BerkeleyDB яка використовує блокування – через це підходить не під всі системи)і в вигляді файлової системи (FSFS–файлова система поверх файлової системи. Використовується по замовчуванню)

Об’єкти в сховищі ініціалізуються двома параметрами – іменем файлата номером ревізії(починається з 0. При чому 0 – це пцста директорія).

**Оперативна ревізія** – ревізія чи діапазон ревізій, до яких має примінитися команда.

svnlog -r 199:230 <http://some.path> (svnlogдо діапазону 199-230)

**Стержнева ревізія** – ревізія яка вказується в додаток до ссилки на об’єкт. Використовується для вирішення неоднозначності, коли в діапазоні ревізій 2 області можна вважати однією історією розвитку об’єкта.

svn log -r 29:33 http://some.path/file.txt@32

svn log -r 29:33 http://some.path/bar.txt@34

Рекомендується в якості стрижневої ревізії вказувати якомога пізнішу версію ревізії стану об’єкта оскільки однозначно ланцюжок станів об’єкту відслідковується тільки назад, а не вперед.

Робоча копія – локальна копія частини даних зі сховища. Мінімальна одиниця, яку можна скопіювати – директорія. Окремо скопіювати файл як робочу копію – не можливо.

Робота в SVNвідбувається за рахунок транзакцій яким притаманна атомарність і ізоляція. Атомарність гарантує, що зміни відбуватимуться однією транзакцію і в випадку збоїв система не залишиться в напівзміненому стані – відбудуться або всі зіміни, або не відбудеться жодної. Ізоляція гарантує, що під час зміни системи інші користувачі не будуть бачити напівзмінену систему в ході операції. Проте, дані властивості не гарантуються в локальній копії – в ній систему може перебувати в проміжковому чи заблокованому стані.

Рекомендується в кореневому каталозі створювати як мінімум 2 директорії:

Не обов’язкове йменування, але це як правило хорошого тону**/.**

**Trunk (тіло проекта)\** основний каталог коду

**Branches (вітки)\** коли треба відійти від основного коду для якихось спроб внесення змін\експерементів.

**Tags (теги, ярлики)\** вітки які мають закінчений характер.

Недоліки SVN – проблеми з перейменуванням файлів, коли в даний момент йде їх зміна, папка svnв кожній директорії, неможливість видалення даних з сховища.

**Команди\послідовність SVN:**

Створюємо папку на локалі і створюємо репозиторій на сервері.

**cd шлях\_до\_локальної\_папки** і в папці:

**svn checkout** https://github.com/PetroAndriets/newSVN\_Demo/trunk - отримання репозиторію - конкретно папки trunk.

Після цього в локальній папці створиться папка trunk.

Виконуємо **cd trunk** в консолі тим самим переходимо в цю папку, яка буде робочою. Бачимо, що створилася скрита папка svn в якій і буде зьерігатися вся історія змін.

Створюємо папку\файл в локальній папці проекту. Потім виконуємо додавання (**svn add testText.txt**) цієї папки\файла в консолі аби вказати системі контролю версій, що ми внесли зміну в структуру проекту створишви цей файл. (після цього ми побачимо як піктограмка на папці\файлі зміниться зі знака питання на плюс)

Робимо якісь зміни з фалом, зберігаємо їх.

Вводимо команду **svn status** (рекомендується вводити її частіше між кроками аби бачити, які зміни відбулися і в якому стані проект). Команда покаже (A testTest.txt), що є папка\файл, які не мають копії на репозиторії сервері, тобто не закоммічені на нім.

Буква М - означає, що фал було змінено. Modify.

Далі робимо комміт змін з повідомленням на сервер: **svn commit -"commit message"**

Нас можуть запитати пароль до репозиторія.

Якщо ми робимо комміт файла, який теж хтось змінив, то виникне ситуація "конфлікту" і комміт не пройде. Для вирішення конфлікту потрібно зробити обновлення (команда **svn update**). Після цього в локалній папці буде залито локальну копію зміненого фал в двох екзкмплярах (файл який міняли ми і цей же фай лякий поміняв хтось інший) з приставкою версій \*.txt.r2 - це попередня версія до змін, \*.txt.r3 - файл який змінював інший кодер і третій файл з приставкою .mine - який міняли ми.

Якщо ми хочемо самостійно вирішити цей конфлікт, а не довіряти автоматичне вирішення його самі СКВ, то після **svn update** прописуємо **p.**

Відкривши оригінальний файл, ми можемо побачити хто які зміни вносив. Вносимо зміни на свій розсуд вирішивши конфлікт, зберігаємся.

І вказуємо СВН, що ми вирішили конфлікт:**svn resolved testText.txt.** Після чого з локальноі папки видаляться три додаткових файла і залишиться тільки оригінальний файл.

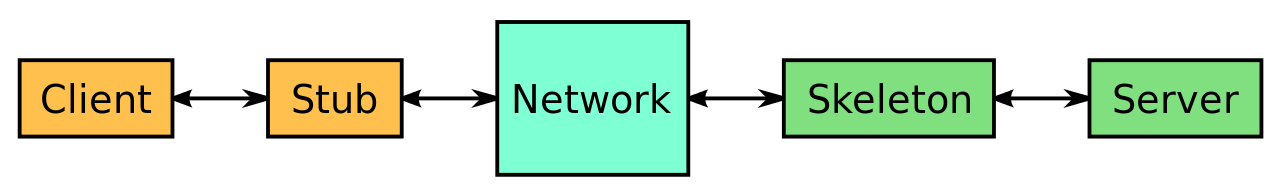
І робимо заключний комміт. **svn commit -"commit message"**

**Локаль** – набір параметрів, включаючи набір символів, мову, часовий пояс, країну і т.д.

**MIME** – стандарт, який описує передачу різних типів через інтернет. Загалом через пошту.

**RPC (RemoteProcedureCall)** – механізм віддаленого виклику процедур в іншому адресному прсторі. Найчастіше – на іншому ПК. Характеристики: асиметричність (одна з сторін є ініціатором), синхронність (виконання викликаючої процедури призупиняється, поки не отримається результат з викликаної процедури).

**RMI (Remote Method Invocation)** – реалізація RPC в Java. (Java RMI).Це програмний інтерфейс виклику віддалених методів в Java на віддаленій ВМ. Відбувається передача параметрів метода на іншу ВМ, сповіщуємо метод про потрібність його виконання, отримуємо результат. Розрізняють клієнтські (які відправили запит) і серверні (які обробляють запит) об’єкти. Вони можуть мінятися місцями по ходу в залежності від ролі. Детально: метод інкапсулюється в заглушку (stub), яка знаходиться на клієнтській стороні. Вона упаковує параметри метода (розгортання параметрів) в потік байтів спеціальним алгоритмом, який не залежить від архітектури платформи. Потім метод заглушки відправляє цю інформацію на серверну частину, далі об’єкт отримувач (скелетон) отримує параметри, шукає потрібний метод, виконує його і повертає пакет даних назад об’єкту заглушці.



**-u root -p -h localhost** – параметризапускуmysql.exe

**Составной SQL запрос** – компонування двох стовбців в якості первинного ключа для запиту.

**Зовнішній ключ** – елемент в таблиці, який містить в собі інформацію для навігації по іншій таблиці.(foreignkey.FK)

**Первиний ключ** – унікальний ідентифікатор записів в таблиці. Ніякі два записи первинного ключа не можуть бути однаковими. (primarykey. PK). Можна сприймати як ідентифікатор кожного рядка таблиці. Не обов’язковий в базі, але рекомендується для підвищення продуктивності БД.

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT **PRIMARY KEY**,

username VARCHAR(20),

create\_date DATE

);

CREATE DATABASE my\_first\_db DEFAULT CHARACTER SET utf8; - створити БД my\_first\_db

**SHOW DATABASES - показує список баз.**

**use bank; -** вибрати ДБ bank для користування.

**source C:\example.sql;** - завантажити файл example.sql

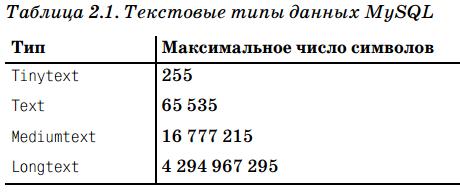
**SELECT NOW();**- дата і час. (Деякі ДБ (Oracle) потребують до команди SELECTnow() доповнення FROMdual; - SELECTnow() FROMdual; де dual–таблиця в якій лише один стовбець dummy)

**CAHR(20);** - строка фіксованої довжинидо 20 символів. (додається пробіл зправа). Максимальна величина 225 символів.

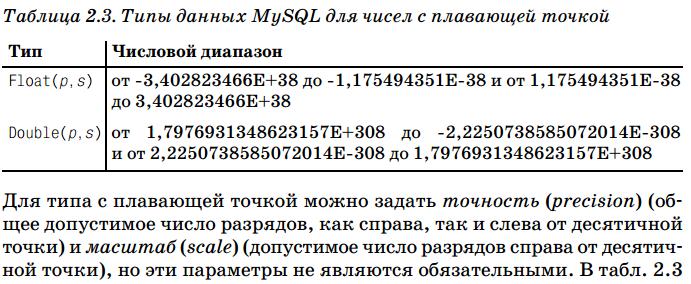
**VARCHAR(20);** - строка змінної довжини. (varchar2 длOracleDB)

**VARCHAR(20) CHARSET SET utf8**; – строка з заданням кодування для стовбця.

**CREATE DATABASE testDB CHARSET SET utf8;** – створення БД з заданням кодування для всієї БД.



Текст, який не помістився в тип просто обрізається.







**CONSTRAINT** – обмеження для таблиці. Вказує БД, який елемент таблиці буде грати роль первинного ключа таблиці.

CONSTRAINT pk\_person PRIMARY KEY (person\_id) – накладається обмеження pk\_person на первинний ключ person\_id.

**Приклад створення таблиці:**

mysql> CREATE TABLE person

-> (person\_id SMALLINT UNSIGNED,

-> fname VARCHAR(20),

-> lname VARCHAR(20),

-> gender ENUM('M','F'),

-> birth\_date DATE,

-> adress VARCHAR(20),

-> state VARCHAR(20),

-> country VARCHAR(20),

-> postal\_code VARCHAR(20),

-> CONSTRAINT pk\_person PRIMARY KEY (person\_id)

-> );

Query OK, 0 rows affected (1.36 sec)

2. CREATE TABLE users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(20),

create\_date DATE

);

Де user\_id INT AUTO\_INCREMENT PRIMARY KEY буде автоматично інкрементувати значення ID юзера як первинного ключа. INT(11). Авто інкремент спрощує процес добавляння юзерів, оскільки сам буде збільшувати значення ID. Первинний ключ також не обов’язковий але рекомендується для покращення продуктивності і архітектури БД.

**Добавлення колонки в існуючу таблицю:**

ALTER TABLE users

ADD email VARCHAR(100)

AFTER username;

**Видалення колонки з таблиці:**

ALTER TABLE users DROP email;

**Зміна властивостей стовбця таблиці (не міняє самих даних):**

ALTER TABLE users CHANGE username user\_name VARCHAR(30);

**Вставка значень в таблицю:**

INSERT INTO users (user\_name. email, create\_date)

VALUES('User', 'user@mail.com', '2015-09-28');

**Показати отсанній інкрементований АйДі для данної сесії:**

SELECT LAST\_INCREMENT\_ID();

**DESC person;** - відобразити таблицю person;

**EXPLAIN users**; - вроді робить ир саме, що й DESC.

**DROP TABLE users**; - видаляє таблицю users.

**SELECT NOW() - показує поточну дату і час.**

SELECT

**GitHub (**розприділена система керування версіями. Тобто є звязок між всіма в команді, а не тільки між окремо кожним і сервером, на відміну від централізованиої SVN**):**

Файл **README.md -** відображається одразу на сторінці репозиторія, тому в ньому можна давати якусь інформацію іншим людям.

Git копіює тільки зміни в файлах, а не весь файл з кодом, наприклад.

**git** - показує команди які можна використовувати.

**git config --global user.name "Petro Andriets" -** задає ім’я користувача гіта. Параметр --global вказує те, що налаштування будуть збережені на постійно.

**git config --global user.email "web.it.www@gmail.com" -** задає електронну пошту користуувача.

Параметри config user.name і user.email обов’язкові, оскільки в репозиторію будуть відображати, хто саме вносив зміни в проект.

**git config --list** - показує налаштування вашого гіта.

**git init -**  ініціалізує поточну папку як гіт проект.

**git clone** https://github.com/PetroAndriets/newGitTest.git - клонує репозиторій newGitTest на ПК.

**git status** - показує чи є і яка різниця між вашою локальною версією проекта і версією на репозиторії в яку могли внести зміни інші люди. Чи змінами які зробили ви, але ще не зробили add цих змін. Потрібно перевіряти зміни, які позначаються зеленим і червоним кольором.

**git add** - додає вказаний файл в локальний проект. (після чого можна робити комміт) Приклад: git add index.html. **git add -A** додає всі файли з проекту. Після змін треба зробити add інакше закрммічено буде попередня версія змін, а не та, яку ви зробили на даний момент.

**git commit -m** "added index.hmtl" - робить комміт. Комміт відбувається в локальний проект. Ключ **-m** дозволяє добавляти повідомлення до комміта.

**git push** - вигружає ваш комміт на репозиторій.

**git pull** - вигружає з репозиторію всі файли, які було змінено. Потрібно бути в папці локального репозиторія для запуску команди. Зазвичай, це перша команда, яку роблять зранку аби завантажити всі зміни. :)

Є 3 області роботи з Git - репозиторій, локальна папка і проміжкова область де файли готуються до pull на репозиторій .Тобто ми вигружаємо проект\зміни з репозиторія в локальну папку, робимо зміни, потім переносимо ці зміни в проміжкову область (git add) і робимо комміт вигружаючи зміни в репозиторій (git commit).

status \ add \ commit \ push

**git branch** - показує в якій вітці ми знаходимося.

**git branch** **newBranch** - створуює нову вітку з імемнем newBranch.

**git checkout newBranch** - перемикається на вітку thisBranch.

**git checkout master -** перемикає на вітку мастер.

**git merge newBranch** - змішує зміни віток мастер і newBranch.

git log --graph

**UML** - діаграми класів. Дозволяють зображати відношення між класами і їх екземплярами. Слугують для декомпозиції (моделювання) предметної області.