I. Въведение

1. Какво е GitHub?

GitHub e уеб-базирана услуга за разполагане на софтуерни проекти и техни съвместни разработки върху отдалечен интернет сървър в т.нар. хранилище (software repository). Базира се на Git системите за контрол и управление на версиите. Услугата може да бъде както платена за частни проекти, така и безплатна за т.нар. проекти с общодостъпен код, като и в двата случая потребителите могат да ползват всички възможности на услугата. Към 2016, GitHub се счита за най-популярния сайт за разполагане на съвместни проекти с общодостъпен или наречен още отворен код.

Сайтът предоставя функционалност за споделяне на софтуерни програми и за работа с тях, като например индивидуално участие на всеки програмист в групови разработки, както и възможност за наблюдение върху работата на всеки участник в екипа.

На GitHub се хостват готови и разработващи се проекти с отворен код, които са свободни за преглеждане, както и работа по тях, което се прави като repository-то се копира на собствения провил. Също така GitHub може да се използва като социална мрежа – има функционалности за следване на потребители и repository-та.

Друга интересна функционалност, която предлага е безплатен хостинг на сайт за всеки проект. Тъй като сайтът е статичен не може да се очаква прекалено много от него. Той може да се направи с чист HTML или използвайки static site generator технологията Jekyll. Jekyll се използва за създаване на статични сайтове и блогове. Той не разчита на база данни и синтаксиса му за писане на постове е Markdown. Минусите които произлизат от това са, че няма динамично съдържание (може да се използват 3d party услуги – например Discus за коментари), различен и по-малко интуитивен начин на администриране от известните CMS системи като Wordpress. Плюсове – безплатен хостинг, висока скорост на зареждане.

Повече за GitHub Pages тук:

<https://pages.github.com/>

GitHub поддържа също така и други услуги:

1. Gist – сайт от типа pastebin, който дава възможност на всеки да съхранява текст или програмен код за опреден период от време в отделно хранилище за данни или сайт, който се достъпва чрез Git-хранилище;
2. Speaker Deck – отново уеб-базирана услуга за разполагане и споделяне на презентации;
3. Gauges – платформа за анализ на интернет трафик и обмен на данни.

Софтуерът, който поддържа GitHub е написан използвайки Ruby on Rails – софтуерна рамка за уеб-приложения и програмния език Erlang.

GitHub предлага хостинг на частни проекти като цената е в диапазона от 7 щатски долара на месец за пет хранилища. В отделни случаи GitHub може да бъде лицензиран за корпоративно ползване на частни сървъри с изградена защитна стена (firewall).

GitHub предлага своето стандартно GUI приложение за сваляне (Windows, Mac, Linux) директно от своя уебсайт.

Предоставяйки уеб интерфейс, автоматично git става достъпна за повече хора. Всеки може да си създаде свободно профил и в него да създаде колкото желае публични хранилища безплатно. Единствено за частните хранилища, които не са публични, има месечна такса.

Github също има много добра документация: [https://help.github.com](https://help.github.com/). Освен това имате директно възможност да работите в тестова среда учейки се.

Много хора свързват Github единствено с дейност, която е свързана с програмиране. Истината е, че сайтът и като цяло git системата се използват за много различни цели освен добавянето на програмен код, например документация, списъци с полезни сайтове, книги и др., закони на държави (bundestag) и още много други категории.

2. Повече за Системите за контрол на версиите

Система за контрол на версиите (на английски: Version control system) е механизмът, по който се управлява работата по даден софтуерен проект. За да се улесни разработката на софтуер са създадени специални системи, които намаляват неудобствата при съвместна работа на много хора върху един проект.

Когато по един проект с множество файлове работят много хора, едно от нещата, което би било трудно без такава система, е да се съобразяват промените по различните части на проекта от всички участници. Система за контрол на версиите се ползва за записване на промените по даден файл или колекция от файлове. Тя позволява да се запази история на промените, да се върне предишна версия, да се добави описание на промените (като например защо и на какво е извършена промяната и други).

Системата за контрол на версиите се развива през годините, като в началото й стои локалната система за контрол на версиите (като RCS), при която файловете са били на локалния компютър на разработчика и само той е работил с тях.

След това се появява централизирана система за контрол на версиите (CVS, Subversion), която позволява много потребители да работят по един проект. Проектът се намира на един централен компютър, с който те осъществяват връзка.

Накрая се появява децентрализирана система за контрол на версиите (Darcs, Bazaar, Git, Mercurial, Fossil, Veracity), която решава най-големия недостатък на другите две – в случай на повреда в централния или локалния компютър, всички данни на проекта могат да бъдат загубени, тъй като се намират на едно физическо място. При децентрализираната система за контрол на версиите, проектът се сваля и копира локално на компютъра на всеки участващ потребител, като така се създава бекъп. В случай на необходимост, проектът може да се възстанови от локалното копие на някой от потребителите.

Система за Контрол на Версиите се използва както за поддържане на кода, така и за документацията и конфигурационните файлове по даден проект. Практически всякакъв тип компютърни файлове могат да бъдат поставени под контрол чрез VCS. Ако например графичен дизайнер има нужда да запази всички версии, варианти или стадии на изображение, или графично оформление, то би било добра практика да използвате такава VCS. Най-простата употреба и обяснение на VCS е ,че потребителят би могъл лесно да се върне към предишна работеща версия на документ, по който работи в случай, че нещо се обърка.

VCS предлагат следните функционалности:

Сътрудничество – група от разработчици, дори и да се намират на различни географски местоположения, могат да работят по един и същ набор от документи или файлове без да пречат един на друг.

Архивиране и Възстановяване - Файловете се поставят в хранилището на проекта след редактирането им и след това можете да се върнете към всеки един момент от развитието на тези файлове.

Синхронизация – Дава възможност на членовете на екипа да споделят документи и да могат да разполагат с най-актуалните им версии.

Отмяна – В случай, че нещо сериозно се обърка ,системата дава възможност да се върнем към последната „ работеща“ версия.

Следене на Промените – Всяка промяна по файловете в хранилището се асоциира с пореден номер и се запазва в история на промените. Когато файла бива променян, може да се добави и кратко обяснение за промените, тяхната необходимост и проблемите, които те решават, което обяснение се запазва заедно с поредния номер в историята на промените на VCS, а не в самия файл. Това прави лесно проследяването на развитието на документа във времето.

Управление на промените – промените могат да бъдат инспектирани, обсъждани, одобрявани или отхвърляни по необходимост, като при желание състоянието на документа или файла може да се върне такова каквото е било на определен етап от развитието си.

Следене на Принадлежността – VCS запазва името на редактиращия към всяка промяна направена от него.

Разклоняване и Обединяване – Възможност за разклоняване на копие от кода в отделна локация и редактирането му в изолация, като при това промените му се следят отделно. По-късно е възможно сливането на кода обратно с друго разклонение.

Непрекъсната Интеграция – възможността, която предлагат системите за контрол на версиите, развитието на даден софтуерен продукт да се раздели на малки части (промени), дава възможността да се провеждат операции, тестове и проверки върху всяка последователна промяна, в непрекъснат стил.

3. Повече за Git

Git е децентрализирана система за контрол на версиите на файлове. Създадена е от Линус Торвалдс за управление на разработката на Linux. Поради нуждата да се контролира огромната база от код на Linux ядрото, основна цел при разработката на Git е била бързината.

Всяка локална Git директория е хранилище с пълна история и възможности за следене на версиите. Това прави Git независим от мрежови връзки към централен сървър.Част от проектите, които използват Git са Linux Kernel, KVM, Bacula, CakePHP, Drupal, FFmpeg, GNOME, phpMyAdmin, Ruby on Rails и много други.

Git е безплатен софтуер и се разпространява под GPL лиценз версия 2.

Git до голяма степен е вдъхновен от BitKeeper и Monotone. Дизайнът е силно повлиян от опитът на Торвалдс с файлови системи. Първоначалният му замисъл е да създаде платформа от ниско ниво за система за контрол на версиите и върху нея да бъдат създавани инструменти като Cogito и StGIT. В процесът на разработка обаче гит се превръща в самостоятелна система за контрол на версиите, която може да се използва директно.

**Структури от данни**

Гит използва два вида структури от данни:

1. Индекс (познат още като stage или cache), който следи промените между работната директория и предстоящата версия за предаване. Структурата е променлива.
2. База от обекти, които не могат да бъдат променяни. Към нея може само да се прибавят нови обекти и те биват четири вида:

* Обект от тип blob, е съдържанието на файл. Обектът не се ангажира да пази метадата подобно на файловете.
* Обект от тип tree, е еквивалентът на директория. Съдържа списък от имена на файлове и съответстващия му blob или друг tree обект описващ под-директория. Представлява моментното състояние на проекта.
* Обект от тип commit, свързва tree обекти за да формира история на промените. Съдържа името на tree обекта, време на промяната, съобщение описващо промяната и имената на предишните обекти от тип commit, ако има такива.
* Обект от тип tag, е контейнер за препратки към друг обект и може да съдържа допълнителна метада свързана с друг обект.

Индексът служи като свръзка между базата с обекти и работната директория.

Всеки обект е уникално идентифицира със SHA-1 хеш на съдържанието му. Гит изчислява този хеш и го използва за име на обекта. Обектът се поставя в директорията отговаряща на първите два символа на хеша. Остатъкът се използва за име на самият обект.

Гит съдържа всяка версия на файл като уникален обект от тип blob. Връзката между тези обекти може да бъде като се разглеждат tree и commit обектите. Нововъведените обекти се съхраняват компресирани, но тъй като се компресират всеки сам по себе си това може да доведе до използване на много дисково пространство. За да се избегне това обектите могат да се комбинират в пакети, които използват делта компресия.

**Характеристики**

**Силно застъпена поддръжка на нелинейно разработване** Гит поддържа бързо разклоняване и сливане, както и необходимите инструменти за визуализиране и навигация в нелинейна история на промените. Разклоненията в гит са просто връзка до един commit. Този commit от своя страна има родител и може да бъде пресъздадена цялата структура.

**Разпределено разработване** Подобно на други системи за контрол на версиите, гит дава на всеки разработчик локално копие на цялата история на проекта. Промените се копират от едно такова хранилище до друго и се третират като допълнителни разклонения, които могат да се сливат също както локалните разклонения.

**Съвместимост със съществуващи протоколи** Хранилищата могат да бъдат създавани чрез HTTP, FTP, rsync, SSH или гит протокол. Гит може също така да емулира CVS сървър. Subversion може да се използва директно благодарение на git-svn командата.

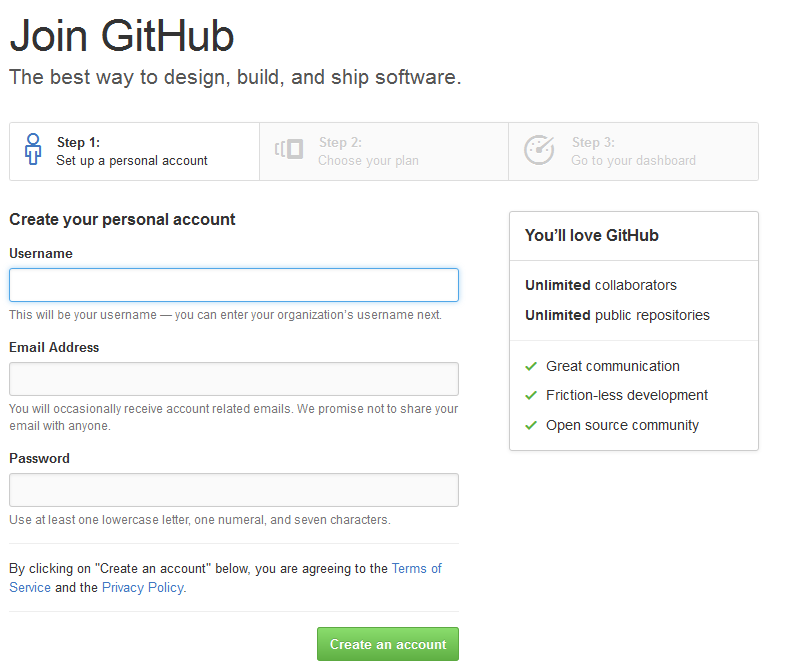
**Ефикасно справяне с големи проекти** По думите на Торвалдс гит е много бърз, без значение от мащаба. Тестове направени от Mozilla показват, че гит е няколко порядъка по-бърз в сравнение с други системи за контрол на версиите.

**Криптографска проверка на историята** Историята в Гит се записва по такъв начин, че идентификацията на определена версия зависи от всички предишни версии. Структурата наподобява хеширано дърво, но с допълнителна информация към разклоненията и листата.

II. Работа по проекта

Регистрация в GitHub

<https://github.com/join>



На началната страница след като влезете в профила си, можете да видите кратки, но основни обяснения за това как да използвате платформата. Препоръчвам да се прочетат:

<https://help.github.com/articles/set-up-git/>

<https://help.github.com/articles/create-a-repo/>

<https://help.github.com/articles/fork-a-repo/>

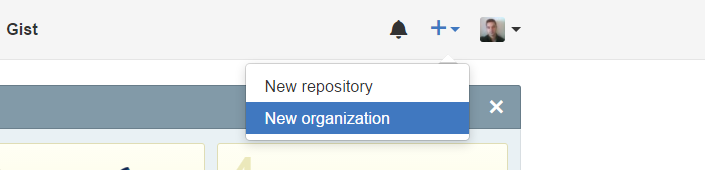
<https://help.github.com/articles/using-pull-requests/>

<https://help.github.com/articles/be-social/>

Началната страница може да се настрои, така че да показва само неща, които ви интересуват.

Създаване на организация

<https://github.com/organizations/new>

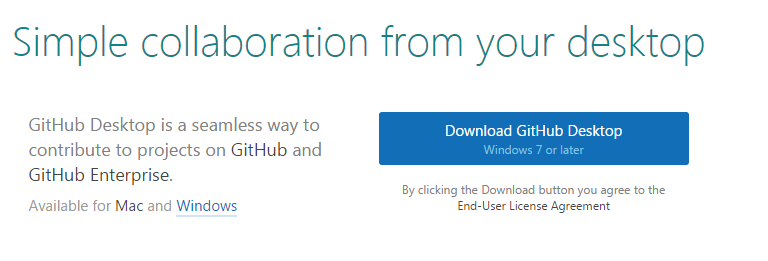


Във връзка с проекта създадох организацията TU Sofia в GitHub:

<https://github.com/TUSofia/>

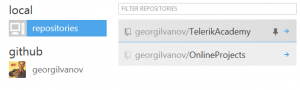
Изтегляне на GitHub Desktop

[http://windows.github.com](http://windows.github.com/)



### Създаване на Хранилище(repository)

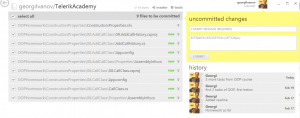
Отворете GitHub for Desktop. В лявата страна на интерфейса може да видите локалните репозиторита на компютърът ви и тези от вашия гитхъб акаунт.



На картинката се виждат хранилищата на компютърa. Мога да вляза във всяко, там гит автоматично засича какви промени съм правил, докато съм работил по файловете. Създаването на локално хранилище става като натиснете бутона +add. Тук трябва да обърнете внимание, че когато изберете директория вашето хранилище ще бъде създадено в нея и всички папки и файлове в нея и поддиректориите ще бъдат добавени към него. Добре е всяко хранилище да има файл README, където да обясните с едно-две изречения какво има в него. Написаното в него се показва, когато някой кликне върху вашето репоситори в гитхъб, така че противно на очакванията – има смисъл да го пишете и наистина някой ще го прочете.

### Commit & Sync

Вече знаете как да създадете собствено хранилище, сега е време да го качим и в гитхъб.



След като сте работили по файловете си можете да влезете в хранилището си и да пристъпите към следващата стъпка – commit. Това означава да добавите файловете по проекта към локалната база данни, където се съхранява вашият проект. Промените по файловете се виждат в ляво. Ако отворите някой ще видите промените по него. В зелено са маркирани добавените редове, а в червено изтритите. Можете да избирате кои файлове да бъдат къмитнати, заглавието на къмита е задължително – постарайте се да отразява промените направени по файловете, при нужда добавете и описание. След като сте добавили промените в локалното хранилище следва да ги синхронизирате с хранилището в гитхъб. Това става чрез бутона sync. Вече файловете навсякъде са еднакви и всички, които работят по този проект имат достъп до най-актуалната версия. В history се виждат всички направени къмити и какво засягат. Възможно е да върнете промените като изберете къмит и натиснете revert commit, а ако сте направили къмит, но още не сте го синхронизирали с roll back this commit се връщат всички несинхронизирани къмити.

### Clone vs Fork vs ZIP

Има няколко начина да изтеглите repository от гитхъб. В горната част на екрана на repository има бутон ZIP. Ако изберете него ще изтеглите като архив всички файлове съдържащи се в хранилището. Те няма да се асоциират по никакъв начин с GUI на Гитгъб или вашият профил. Избирайки Clone in Windows вие ще изтеглите репозиторито и то ще се запази в default директорията за проекти зададена в GUI програмата. След това ще можете да работите по същият начин както с локалните хранилища, които вие сте създали, няма да можете да синхронизирате, ако нямате права. Другият вариант да допринесете за даден проект, contribute или на български – да съучастничите е най-добре да форкнете(fork) репозиторито. Това означава да създадете собствено разклонение (нещо като личен branch), който се вижда във вашият профил и да имате локално копие на компютъра си. По този начин вие можете да работите по проекта и когато свършите даден модул давате заявка за Pull.

### Pull и fork sync

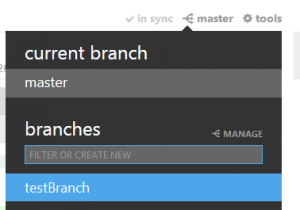
Когато pull-нете вашата работа към главният бранч подавате заявка за присъединяване на вашият код към главният бранч. В нея се вижда историята на вашите къмити, коментарите и допълнително описвате какво е направено. Всеки пул може да се разглежда и обсъжда в секцията Pull Requests.

Докато работите на вашето форкнато копие е възможно да се появят промени, които да искате да добавите към проекта си. За целта трябва да отворите шел в директорията на fork-а(дясно копче върху проекта open shell here). Въвеждате следните команди:

git remote add upstream [https://github.com/user/project](https://github.com/user/project.git) git fetch upstream git merge upstream/master

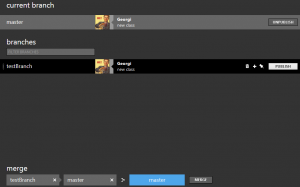
Първата команда добавя отдалечено репозитори с име upstream от даденият адрес(можете да го вземете в web страницата на проекта). Втората взема разликите между вашият бранч и там откъдето сте го форкнали. Последната слива вашият master с upstream.

### Branch



Създаването на клон(branch) е изключително важна част от version control-а. Бранчът представлява копие на главното(master) репозитори. Всеки път, когато искате да започнете работа по даден бъг или подобрение в програмата е препоръчително да създадете нов бранч, който адресира точно тази промяна. Създаването и мърджването на бранчове през GUI е много лесно.

Когато сте в проекта си, натиснете бутона в дясно от sync и ще ви излезе панел с всички бранчове. Можете да ги сменяте с един клик. На първата картинка се вижда, че в момента съм в клона master, като имам и testBranch, в който имам код напълно независим от master. В manage се контролират всички локални бранчове. Можете да изберете откъде да създадете нов бранч, да публикувате вече създаден от вас бранч(така по него ще могат да работят и други) и да събирате бранчове. В долната част на втората картинка съм добавил с drag’n’drop двата бранча и при натискане на бутона merge ги събирам в master.



### Основни Git Команди

**init** Използва се за първоначално инициализиране на проект. Проектът се създава в текущата директория.

**commit** Записване на промените по файловете. Когато промените съдържанието на даден файл, намиращ се в директорията на проекта, Git автоматично ще засече тази промяна. За да се запише самата промяна обаче, ще трябва да направите commit заявка. Ако ползвате графичен Git клиент, Git ще ви покаже, че е била извършена промяна и ще ви попита, дали искате да направите commit (запис) на тази промяна.

**pull** Заявка за изтегляне/качване на промени от едно хранилище/клон в друго хранилище/клон.

**merge** Сливане на промените в един клон или сливане на отделни клони. След като промените по файловете, в дадено разклонение, са записани, можете да създадете pull заявка за сливане. Тази pull заявка ще е в статус изчакване за одобрение. След одобрение, посоченият клон ще бъде слят, използвайки git merge, с друг избран клон.

**clone** Използва се за сваляне на съществуващ проект.

**add** Използва се за първоначално добавяне на файлове към проекта или тяхното променяне.

**status** Използва се за проверка на състоянието на проекта и файловете в текущото разклонение.

**diff** Използва се за визуализиране на промените във файловете, които не са записани в индекса (unstaged).

**reset** Използва се за изваждане на файлове от индекса (unstaging) с цел да не влязат в следващият commit.

**rm** Използва се за перманентно премахване на файл от индекса.

**stash** Използва се за запазване моментното състояние на индекса с цел да се довърши по-късно и връща състоянието на последния commit.

**branch** Изброява, създава и управлява разклоненията на проекта.

**checkout** Сменя временният контекст(branch), в който се работи

**log** Използва се за проследяване историята на промените в проекта.

**tag** Използва се за отбелязване на commit, който трябва да се отличава от другите. Всяка окончателно завършена и одобрена промяна се слива с master и представлява различна версия на проекта. Промяната може да се надпише с различен таг, например започвайки с v. 1.0.0.

**fetch** Използва се за сваляне на нова разклонения от отдалечени хранилища

**push** Качва промените на отдалечено хранилище.

**remote** Изброява, добавя и премахва псевдоними на отдалечени хранилища.

### Примери

git pull origin master – проверява дали няма нови промени по сървъра и ги изтегля

git status – проверява статуса на файловете и най-вече дали има такива за commit или add и ги показва

git add /пътят до някой файл или „\*“ – добавя променени или нови файлове или папка за качване в git

git commit –m ‘Коментар‘ – маркирва всички промени и им добавя коментар

git push origin master – качва всички маркирани промени

# Използване с Eclipse

1. За да включим останалите от екипа си към repository и да commit-ваме заедно по прокта си правим следното: В repository-то, което ще споделяте, от дясно на профила ви, под менюто + , избирате New collaborator. Въвеждате потребителското Git име на ваш съотборник и го добавяте. Повтаряте процедурата с всички съотборници.
2. Да предположим, че това ви се случва на работното място или в университета, и вечерта се прибирате у дома. Първото, което трябва да направите е да си дръпнете проекта от GitHub в Eclipse, за да можете да работите до късно по него. Отваряте си Eclipse -> Import, на Sеlect an import souрce-> избирате Git-> Project from Git На следващата стъпка имаме: Select a Repository Source->Избираме-> Clone URL в следващия прозорец въвеждаме URL-то на Git repository-то, което желаем да свалим -> Next-> Next и готово, имаме последна версия на проекта, локално при нас. Този импорт на проект от Git e възможен не само при екипен проект, а винаги когато нещо ни хареса в Git и желаем да си го свалим и разработваме при нас.
3. Правим промени по кода и другите ни съотборници правят промени, следваща стъпка - Synchronize Workspace- синхронизация. От Project Explorer даваме десен бутон на проекта Team ->Synchronize Workspace. Отваря се перспектива, която показва файловете за ъпдейт(файлове, които са променени, но не на вашата машина), за commit(вашите промени), конфликти(ако двама са роботили по един файл) и т.н
4. Ъпдейта на актуалната версия на файловете се случва в персективата за Синхронизация или Git перспективата или дори през Project Explorer, чрез десен бутон на файла -> Pull. Pull се използва за сваляне на промените от отдалечено хранилище и се прави опит да се слеят заедно с локалното.
5. Commit на вашите промени и Push to Upstream синхронизира вашите промени по кода, като ги приобщава към последната актуална версия на кода.

На практика GitHub e едно динамично хранилище, което ви прави независими от външни носители. Във всеки един момент можете да проследите направените промени по даден клас. В Eclipse, това става, като дирекно върху сорс кода давате десен бутон-> Comare With. Имате възможност да сравните, както с предишна версия, с последния Commit, както и в локалната история- по часове и дати.

# Пакет за студенти на GitHub

Нещо интересно и полезно, което открих е GitHub Education Pack. Това е безплатен ъпгрейд на регистрацията, който GitHub предлага на ученици и студенти, с до 5 private repositories. GitHub Education Package върви в пакет с безплатни услиги от партньорски организации на GitHub, включващи:

* AWS Educate
* Bitnami
* CrowdFlower
* DigitalOcean
* DNSimple
* GitHub
* HackHands
* Microsoft DreamSpark
* Namecheap
* Orchestrate
* SendGrid
* Stripe
* Travis CI
* Unreal Engine

Повече информация [тук](https://education.github.com/pack).

За да получите GitHub Education pack трябва да попълните тази [форма](https://education.github.com/discount_requests/new). За да докажаете че сте студент снимайте студентската си книжка и прикачете снимката. Също така трябва да опишете за какво ще използвате GitHub по време на следването си.

III. Резултати

1. Създадена е GitHub организация която ще бъде използвана за разработване, поддържане и подобряване на групови проекти по време на обучение в ТУ София.

<https://github.com/TUSofia>

2. Създадена е документация съдържаща основна информация за Git, GitHub и Системите за контрол на версиите, и как да бъдат използвани. Документацията е под формата на Wiki статии и е достъпна на <https://github.com/TUSofia/vavedenie/wiki>

Списък с wiki статии:

* [Видео GitHub](https://github.com/TUSofia/vavedenie/wiki/%D0%92%D0%B8%D0%B4%D0%B5%D0%BE---GitHub)
* [Видео Version Control Systems и Git](https://github.com/TUSofia/vavedenie/wiki/%D0%92%D0%B8%D0%B4%D0%B5%D0%BE-Version-Control-Systems-%D0%B8-Git)
* [Използване с Eclipse](https://github.com/TUSofia/vavedenie/wiki/%D0%98%D0%B7%D0%BF%D0%BE%D0%BB%D0%B7%D0%B2%D0%B0%D0%BD%D0%B5-%D1%81-Eclipse)
* [Използване с Visual Studio](https://github.com/TUSofia/vavedenie/wiki/%D0%98%D0%B7%D0%BF%D0%BE%D0%BB%D0%B7%D0%B2%D0%B0%D0%BD%D0%B5-%D1%81-Visual-Studio)
* [Как да използваме Git?](https://github.com/TUSofia/vavedenie/wiki/%D0%9A%D0%B0%D0%BA-%D0%B4%D0%B0-%D0%B8%D0%B7%D0%BF%D0%BE%D0%BB%D0%B7%D0%B2%D0%B0%D0%BC%D0%B5-Git%3F)
* [Как да използваме GitHub?](https://github.com/TUSofia/vavedenie/wiki/%D0%9A%D0%B0%D0%BA-%D0%B4%D0%B0-%D0%B8%D0%B7%D0%BF%D0%BE%D0%BB%D0%B7%D0%B2%D0%B0%D0%BC%D0%B5-GitHub%3F)
* [Как да качим код в GitHub?](https://github.com/TUSofia/vavedenie/wiki/%D0%9A%D0%B0%D0%BA-%D0%B4%D0%B0-%D0%BA%D0%B0%D1%87%D0%B8%D0%BC-%D0%BA%D0%BE%D0%B4-%D0%B2-GitHub%3F)
* [Какво е Git?](https://github.com/TUSofia/vavedenie/wiki/%D0%9A%D0%B0%D0%BA%D0%B2%D0%BE-%D0%B5-Git%3F)
* [Какво е GitHub?](https://github.com/TUSofia/vavedenie/wiki/%D0%9A%D0%B0%D0%BA%D0%B2%D0%BE-%D0%B5-GitHub%3F)
* [Какво е Version control system?](https://github.com/TUSofia/vavedenie/wiki/%D0%9A%D0%B0%D0%BA%D0%B2%D0%BE-%D0%B5-Version-control-system%3F)
* [Пакет за студенти на GitHub](https://github.com/TUSofia/vavedenie/wiki/%D0%9F%D0%B0%D0%BA%D0%B5%D1%82-%D0%B7%D0%B0-%D1%81%D1%82%D1%83%D0%B4%D0%B5%D0%BD%D1%82%D0%B8-%D0%BD%D0%B0-GitHub)

3. Създадена е GitHub Pages web страница на проекта:

<http://tusofia.github.io/vavedenie/>