**Git и GitHub**

1. **Git**

Git e система за контрол на версиите (Version control system). Git  е механизмът, по който се управлява работата по даден софтуерен проект. Системата за контрол на версиите улеснява съвместната работа на много хора върху един проект. Нарича се още разпределена система за контрол на програмния код (source-control system), както и Software configuration management (SCM).

През 2005 г. Линус Торвалдс (създателят на Линукс ядрото) решава, че системата, която използва за проследяване на промените по кода, не му върши работа. Така на бял свят се появява git.

Основната идея е лесно да могат да се запазват всички промени в сорс-кода и да се проследяват разликите между една или версии. Git позволява включването на неограничен брой хора към даден проект,.

Системите за контрол на версиите обикновено използват едно централно хранилище, в което се съхраняват файловете на проекта. Когато някой започне да работи по този проект, той създава копие на файловете от това хранилище на своята система и работи с тези копия. В процеса на работа потребителя може да изпраща към хранилището направените от него промени и да получава промените, направени по проекта от другите хора в екипа.

След обновяване на хранилището, в него се запазват старите версии на променените файлове, т.е.. при съвместна работа по даден файл се помни пълната история какво точно е било променяно. Всяка промяна се описва, например фикснат бъг, допълнение и др. Всяка една от тези версии може да бъде възстановена при необходимост.

Системата за контрол на версиите също така следи за конфликти – промени на един и същ файл от различните потребители, които ползват хранилището.

Git е най-популярната в света (към момента) система за контрол на версиите, свободна, с отворен код. Работи с локални и отдалечени хранилища.

1. **GIT Hub**

През 2007 г. в САЩ трима американци (Tom Preston-Werner, Chris Wanstrath, PJ Hyett) решават, че могат да използват git системата по още по-добър начин, създавайки [Github.com](http://github.com/) през 2008 г. 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.

Репозиторито може да бъде Remote или Local.

Remote repository – намира се на сървър, на отдалечена машина и всички работят с него

Local repository – копие на локалната машина, което ние променяме независимо от remote repository. Когато сме готови с промените, трябва да ги синхронизираме.

GitHub е сайт за хостване на сорс-код. Той е е изключително популярен през последните години и много полезен при търсене на работа. Чрез GitHub може дори и начинаещите програмисти без опит да представят себе си пред бъдещите си работодатели.

GitHub е хранилище за сорс-код.

Осигурява копие на кода от нашия компютър

Дава възможност за екипна работа на повече души по един и същ проект едновременно

Кодът е общодостъпен и свободен за ползване

Можем да научим добри практики от кода на другите в GitHub

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: документация, списъци с [полезни сайтове](https://github.com/ripienaar/free-for-dev), [книги](https://github.com/vhf/free-programming-books) и др., закони на държави ([bundestag](https://github.com/bundestag/gesetze)) и още много [други категории](https://github.com/explore). Github има над 9 милиона активни потребители към 2015 г. и повече от 10 милиона хранилища.

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.

1. **Работа с GitHub**

**а/ разглеждане на гитхъб на други потребители – не изисква регистрация**

<https://github.com/ivaylokenov>

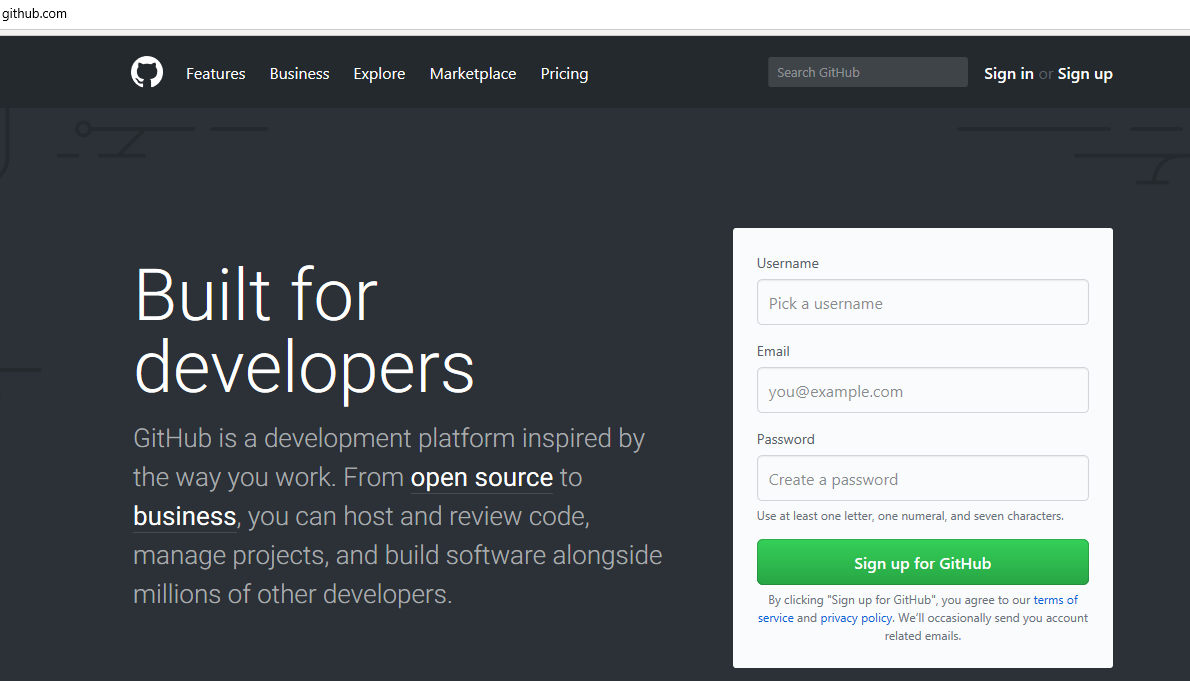
Nikolay Kostov и др.

Можем да разгледаме колко промени /Commit/ има в дадено хранилище, от кой точно е направена съответната промяна, както и каква е точно промяната. Можем да изтеглим архив на дадено хранилище и да го разглеждаме, както и да го позваме както намерим за добре – може да го променим, да го включим в наши проекти и др.

Fork – изтеглям репозиторито в моя гит хъб

Star – звездички, като лайковете във фейсбука

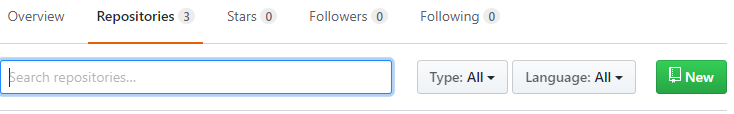
**б/ регистриране на потребител**



**в/ Създаване на хранилище /Repository/от менюто горе в дясно**

Repository е всичко свързано с даден проект – файлове, история на промените, коментари.

Създаваме хранилище на сървъра като в началната страница на github.com изберем Repositories » New



В следващия прозорец описваме хранилището:

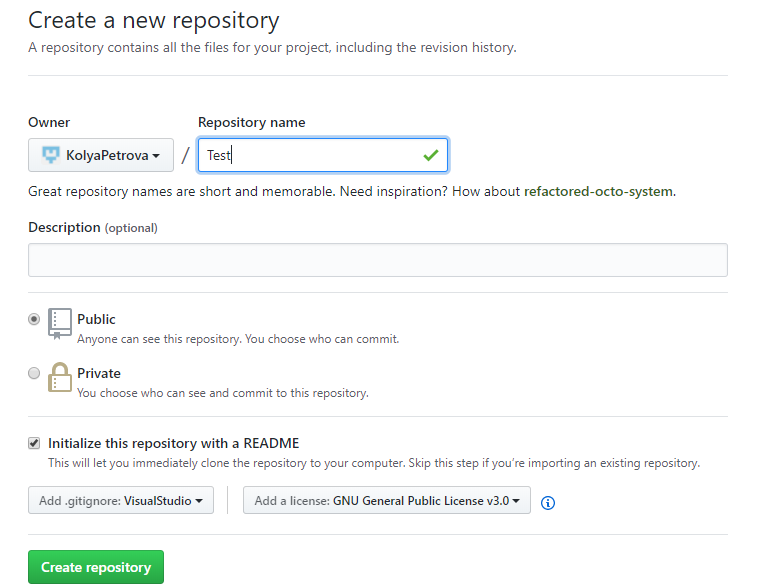
Име, кратко описание,

Public – по подразбиране безплатните акаунти създават само публични хранилища. За да имаме права да правим частни, трябва да се плаща месечна такса /7$/.

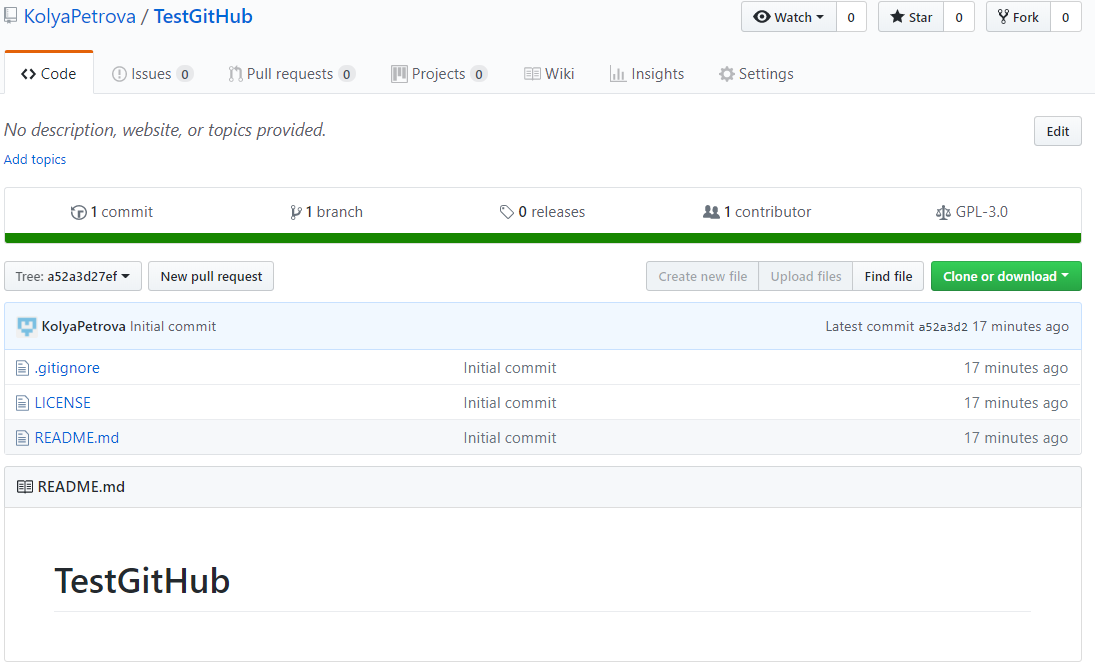
Добавяме ReadMe – поставяме отметка в полето Initialize this repository with a README, с което указваме, че искаме да се добави този файл към хранилището. В него може да впишем важна информация за проекта.

От падащия списък в gitignore можем да изберем какви файлове да се игнорират, например ако пишем на C# ще изберем visual studio и ще се игнорира папка .vs, която е скрита и съдържа файлове, които не са необходими за стартирането на проекта. Ако пишем игри, ще игнорираме ненужните файлове на Unity.

От списък Add a license можем да добавим файл с лиценз GNU General Public License или за обучителни цели използваме MIT лиценз, който означава, потребителят може да прави каквото си иска с нашия код, но ние не носим отговороност за нещо което може да се случи от използванет на кода ни.



След натискане на бутон Greate repository се създава ново хранилище със следните три файла:



**г/ променяне на файлове – всяка промяна се помни в хранилището /commit/**

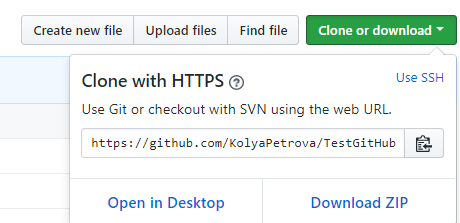
При двойно кликване върху името на файла виждаме съдържанието му. С бутон Edit /моливче/ можем да променяме съдържанието на файла. Съхраняването на промените става с бутон Commit, като преди това описваме какви промени и в кой файл сме направили.

**е/ добавяне на файлове към проекта /Upload/**

Можем да добавим файлове от нашия компютър като ги изтеглим в полето за добавяне на файлове. Тази промяна също се помни като част от историята /нов commit/

1. **Работа с хранилище на локалния компютър**

Локално репозитори се създава с команда Clone или обикновено изтегляне. При изтеглянето се изтегля RAR архив, който разархивираме и работим с него, но той не е свързан с отдалеченото копие. При клонирането вземаме URL адреса и по този начин се създава връзка между локалното и глобалното копие и можем лесно да ги синхронизираме. След клонирането в папката на проекта се създава скрита папка .git, която пази информация за връзката на локалното и отдалеченото репозитори.



За работа с локалното хранилище /за запазване на версиите на сорс-кода, както и за синхронизиране с отдалеченото хранилище/ ни трябва специален софтуер за работа с хранилища. Използваме два вида – с текстов и с графичен интерфейс:

**а/ Git bash – команден интерфейс към Git (msysGit):**

* Изтегляме от [http://msysgit.github.io](http://msysgit.github.io/)
* Инсталираме всичко по подразбиране
* Стартираме Git Bash
* Основни Git команди

Клониране на съществуващо Git хранилище: **git clone [отдалечен url]**

Изтегляне и сливане на промени от отдалечено хранилище: **git pull**

Подготовка (добавяне / избор) на файлове за запис: **git add [файл]** ("**git add .**" добавя всичко)

Предаване (commit) към локалното хранилище: **git commit –m "[вашето съобщение]"**

Проверка на статуса (промените) в локалното хранилище: **git status**

Създаване на ново локално хранилище (в текущата папка): **git init**

Създаване на отдалечено (+кратко име за отдалечен Git URL): **git remote add [remote name] [remote url]**

Изпращане на промени (към отдалечено хранилище): **git push [remote name] [local name]**

**Пример**

Създаваме нова директория: mkdir work

Избираме я за текуща: cd work

Клонираме репозиторито git clone https://github.com/SoftUni/test.git

Проверяваме съдържанието на тек. директоря: dir

Влизаме в папката на хранилището cd test //ако името друго, пишем него

Проверяваме съдържанието на папката на хранилището: dir

Проверка на статус: git status

Редактираме някой файл например отваряме ReadMe и променяме, save

Проверка на статус: git status

Запазваме версия в историята /commit/: git commit -m "changes"

Добавяне на нов файла към репозиторито: git add и името на файла или git add . / за да добавим всички файлове

Запазваме версия в историята /commit/: Git commit –m “Added another file”

Качваме в отдалеченото хранилище: git push или Git push origin master - да го качим там, откъдето сме го изтеглили. Ако пропуснем Origin master, ще се качват във всички бранчове, което в нашия случай е същото като ако посочим бранча, тъй като имаме само един бранч

В сайта се вижда и новодобавения файл. Променяме файла в сайта и комитваме.

Изтегляне на променения в сайта файл на локалното репозитири: git pull

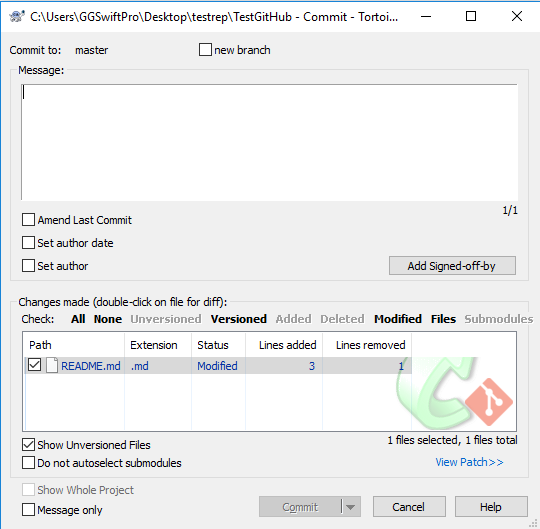
Тестова среда за обучение: <https://try.github.io/levels/1/challenges/1>

**б/ TortoiseGit – графичен клиент за работа с GitHub, трябва да имаме инсталиран Git**

Изтегляне: <https://tortoisegit.org/download/>

Инсталираме, автоматично открива git.exe

Задаваме име и е-майл, с който сме регистррани

Инструментът се интегрира в Windows Explorer и е достъпен с д.б. на мишката върху папка, в която ще клонираме.

Commit – създаване на история на промените, които правив във файлове на локалния компютър

Д.б. върху папката на репозиторито и Git Commit … - отваря прозорец, в който показва в кой файл и къде точно сме правили промени. Добавяме текст в полето Message и чрез Add signed off by добавяме името си и кликваме в бутона Commit. Създава се нещо като Check Point. Файловете на отдалечения компютър не са променени. С Commit само запазваме история на промените.

- променяме readme файла на нашия компютър и Git Commit

- добавяме нов файл Git Commit

- създаваме нов проект на Visual Studio в папката на локалното репозитори и отново Git Commit.

Така ние създаваме 3 промени, но те са само на нашия локален компютър.

Синхронизиране на локалното и отдалеченото репозитори – става в две посоки:

**Отгоре надолу /от отдалечено към локално/ - fetch**

Отдолу нагоре /от локално към отдалечено/ - upload

Синхронизиране отдолу нагоре

С д.б. върху папката и Git Sync – отваря прозорец, в който с Push качваме локалното репозитори на отдалечения сървър. Иска да въведем име /или мейл/ и парола. Тези данни могат да се зададат в настройките на програмата, за да не ги пишем всеки път. След синхронизирането промените вече са на сървъра.

**Синронизиране отгоре надолу**

Правим промяна в readme файла на сървъра и ще синхронизираме локалното репозитори. С д.б. върху папката и Git Sync и в отварения прозорец кликваме в бутон Pull. Промените вече са и на локалния компютър.

**Конфликт**

Честа практика е двама или повече души да работят по един файл (или променим даден файл от хранилището през браузера, или ние работим в две локални копия на едно и също хранилище/. Първият качва промените си стандартно с Push. Когато вторият се опита да качи неговите промени в отдалеченото хранилище, става ясно, че оригиналът, по който той е работил, вече не е същият, и тогава възнква конфликт.

**Пример:**

Променяме отдалечения readme файл и комитваме

Променяме локално нашия readme файл и Git Commit.

Синхронизираме нагоре с push и получаваме съобщение за грешка /конфликт/

**Решаване на конфликта**

Изтегляме отгоре промените /Pull/ при което отново получаваме съобщение за конфликт.

С д.б. върху файла и Edit отваря текстов редактор, в който виждаме двата файла и решаваме кои промени да вземем. В полето отдолу виждаме съдържанието на файла след редакцията. След като редактираме съдържанието. Съхранияваме и маркираме като решен проблем.

Commit – /съобщението показва, че сме решили конфликт/

Синхронизираме като качваме нашия файл в отдалеченото репозитори.

Канбан (на [японски](https://bg.wikipedia.org/wiki/%D0%AF%D0%BF%D0%BE%D0%BD%D1%81%D0%BA%D0%B8_%D0%B5%D0%B7%D0%B8%D0%BA): かんばん) е метод за управление на интелектуалната дейност, който набляга на доставката точно на време, без същевременно членовете на екипа да са пренатоварени. При този подход процесът - от дефинирането на задачата до доставянето на продукта, е изложен на показ за участниците в него, а същевременно разработчиците на софтуера взимат работни задачи от така наречената „опашка“. Канбан може да бъде разделен на две части:

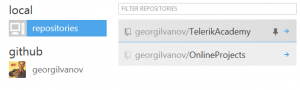
[Канбан](https://bg.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B1%D0%B0%D0%BD) - Система за визуално управление на процеси, която указва какво да се произведе, кога да се произведе и какво количество да бъде произведено.

Методът Канбан - Подход за постепенно, еволюционно подобряване на процесите в една организация.

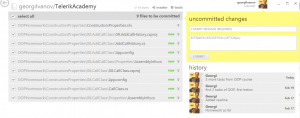
<http://georgi-ivanov.com/how-to-set-up-git-in-windows/>

# How to set up Git in Windows

## Set Up Git

[](http://georgi-ivanov.com/wp-content/uploads/2013/02/navigationGit2.png)Първо трябва да си направите акаунт в [https://github.com](https://github.com/), ако нямате вече такъв. След това трябва да изтеглите и инсталирате native клиента за windows – [http://windows.github.com](http://windows.github.com/). В лявата страна на интерфейса може да видите локалните репозиторита на компютърът ви и тези от вашия гитхъб акаунт. На картинката в ляво се виждат хранилищата на моят компютър. 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.txt, където да обясните с едно-две изречения какво има в него. Написаното в него се показва, когато някой кликне върху вашето репоситори в гитхъб.

## Commit & Sync

[](http://georgi-ivanov.com/wp-content/uploads/2013/02/gitcommit.png)Време да качим хранилището 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 има бутон **ZIP**. Ако изберете него ще изтеглите като архив всички файлове съдържащи се в хранилището. Избирайки **Clone in Windows**вие ще изтеглите репозиторито през gitapp и то ще се запази в default директорията за проекти зададена в 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 собствен клон(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

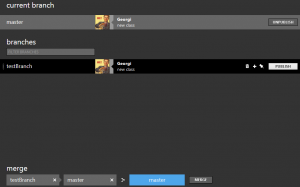
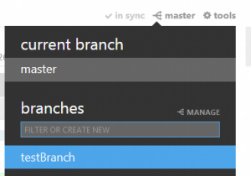
Когато pull-нете вашата работа към главният бранч подавате заявка за присъединяване на вашият код към главният бранч. В нея се вижда историята на вашите къмити, коментарите и допълнително описвате какво е направено. Всеки пул може да се разглежда и обсъжда в секцията **Pull Requests**.

Докато работите на вашето форкнато копие е възможно да се появят промени, които да искате да добавите към проекта си. За съжаление все още няма директна поддръжка от gitapp-a това да се случи. 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.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

[](http://georgi-ivanov.com/wp-content/uploads/2013/02/gitbranchMergePub.png)[](http://georgi-ivanov.com/wp-content/uploads/2013/02/gitbranchPanel.png)Създаването на клон(**branch**) е изключително важна част от version control-а. Бранчът представлява копие на главното(master) репозитори, но част от целият проект и асоцииран с него. Смисълът да го правите е, че ако омажете нещо няма да ревъртвате 100 къмита в главният проект или пък ако работата върви по-бавно няма да бавите другите. 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 gitapp е многолесно. 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.

## Недостатъци

Не може да се тегли/бранчва/форква само една папка. Ако имаме хранилище с няколко проекта и искаме да работим само по един – трябва да теглим цялото хранилище.

Вторият недостатък е че няма безплатни **private**хранилища.

## Защо да използвам GitHub

Това, което им дава предимство пред другите безплатни алтернативи е безпрецедентният контрол върху сорса. Всичко е подредено и систематизирано изключително добре. Всички канали за комуникация са налице. Огромно къмюнити, въпреки, че не е насочено много към технологии произлизащи от Microsoft. Приятен и удобен за работа интерфейс.