



Програмиране на Java

Концепции за контрол на версиите

Как
поддържа
различни
версии на
един и същи
файл?



Как
работим
съвместно
по един и
същи
проект?





—





Защо е необходим контрол на версиите?

Система за контрол на версиите (Version Control System, VCS) е инструмент за управление на набор от файлове (предимно с програмен код), който предоставя три основни възможности:

- Обратимост (reversibility)
- Едновременна работа (concurrency)
- Анотиране (annotation)

VCS: Обратимост

Възможността да се върнете към запазено предишно състояние, което знаете, че е било добро, тогава когато откриете, че някаква модификация, която сте направили, е била неправилна или идеята, която сте имали не е съвсем успешна.





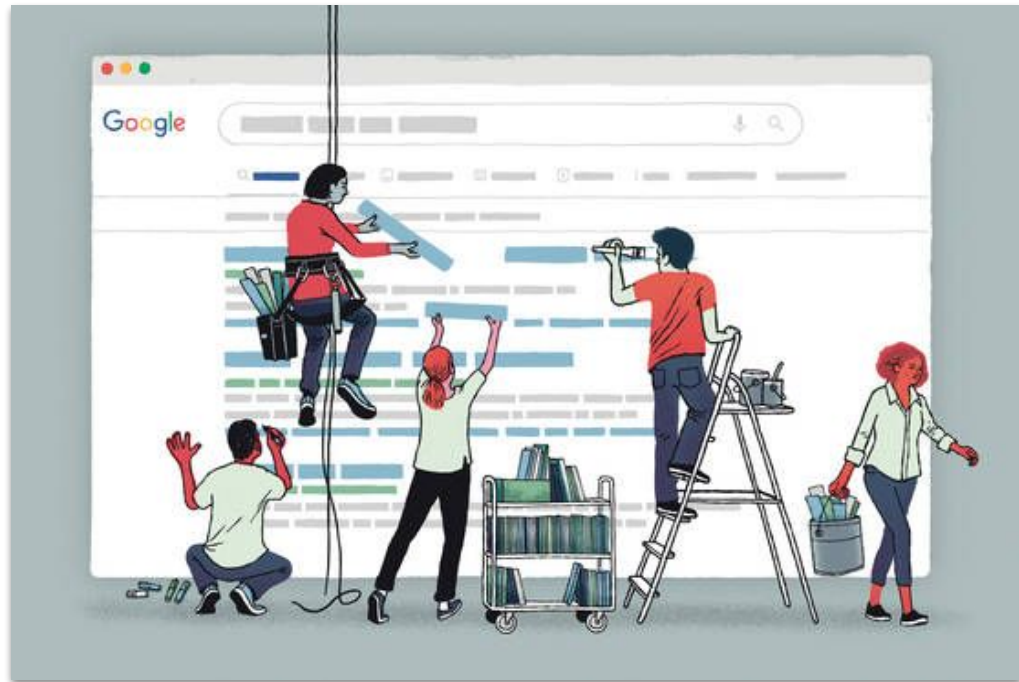
VCS: Обратимост

Запазва се история на промените.

- Застрахова срещу загуба на данни.
- Позволява връщане на предишна версия на файловете при необходимост.
- Дава възможност за справка с версия на файловете назад във времето.
- Позволява отмяна на определени промени направени в миналото без да се губят промените направени междувременно.
- За всеки ред във файла може да се разбере кога, защо и от кого е направена промяната.

VCS: Паралелност

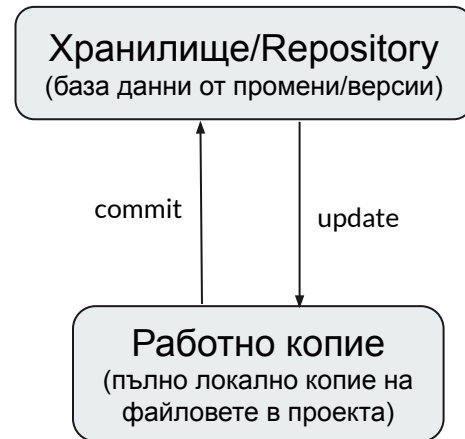
Възможността много хора да модифицират един и същи набор от файлове по едно и също време, при което конфликтните модификации могат да бъдат решени автоматично.



Хранилища и локални копия

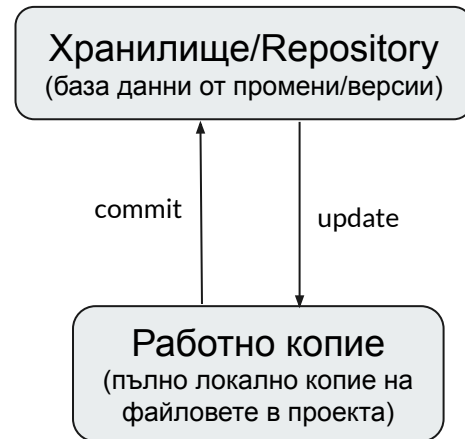
Системите за контрол на версиите използват хранилище (**repository**) и работно копие (**working copy**), върху което се правят промените.

- Работното копие (познато също и като **checkout**) е личното копие на всички файлове в проекта. Може да правите промени по файловете в това копие, без това да се отрази при останалите участници в екипа.
- Когато сте готови с редакциите, се прави **commit** (прилагане) на промените в хранилището.



Хранилища и локални копия

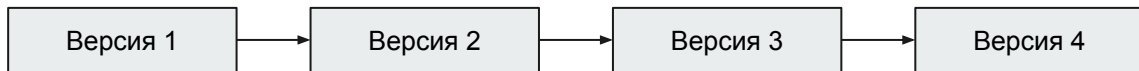
- Хранилището (repository) е база данни с всички промени на проекта. Също така съдържа и историческите версии (**snapshots**) на проекта.
- Възможно е в хранилището да има промени, направени от друг член на екипа, които все още не са приложени върху работното ви копие.
- За да бъдат приложени върху работното ви копие трябва да се направи **update** (актуализация) на работното копие.






История на промените

В най-простият случай базата данни на хранилището (repository) съдържа линейна история: промените са направени последователно във времето.



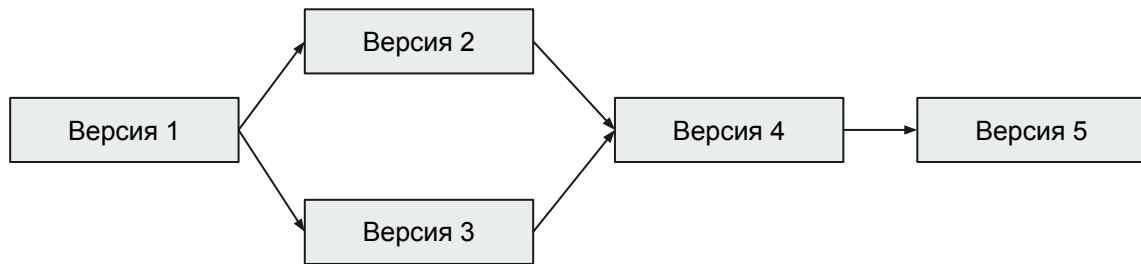
Линия на времето




Най-често това се случва когато само един човек работи по проекта в даден период от време.

История на промените

Когато по един проект работят повече от един човек едновременно са възможни разклонения (**branching**) в историята на промените, които в последствие се обединяват (**merge**) отново.

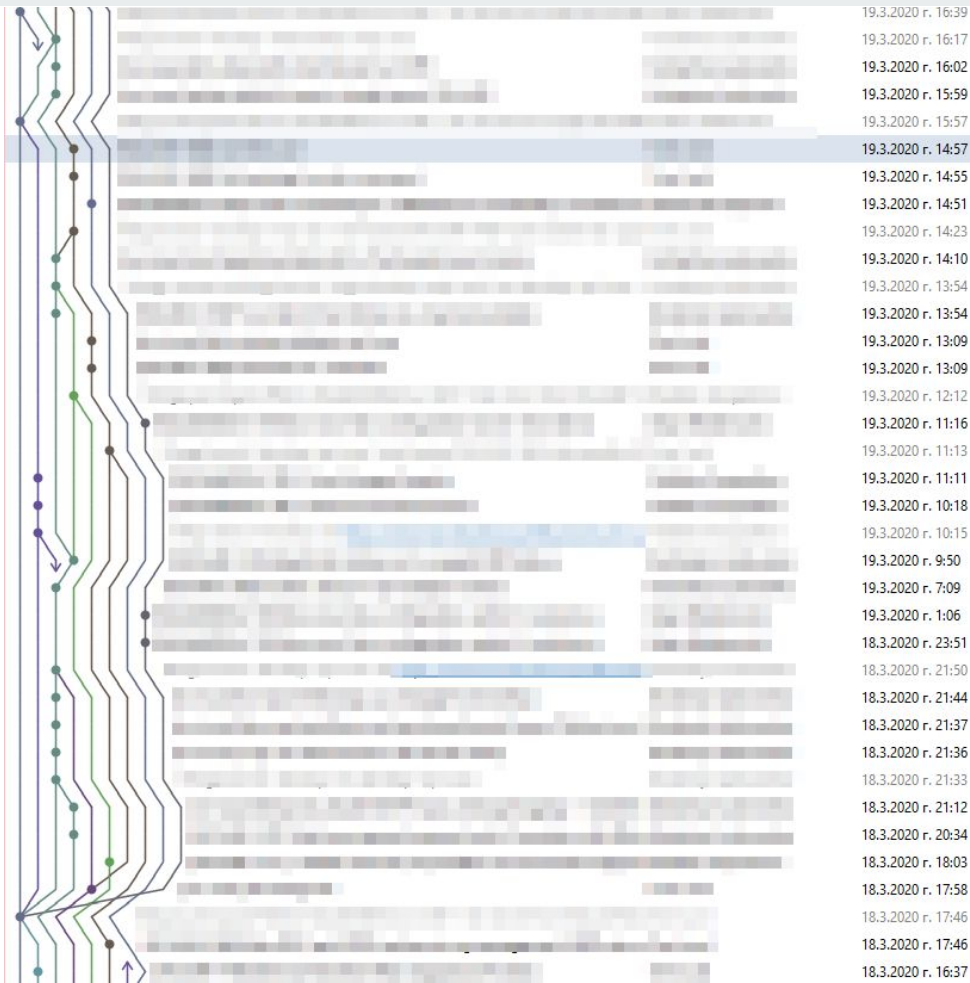


Линия на времето



История на промените

Пример с историята на промените и обединяването им от реален проект.





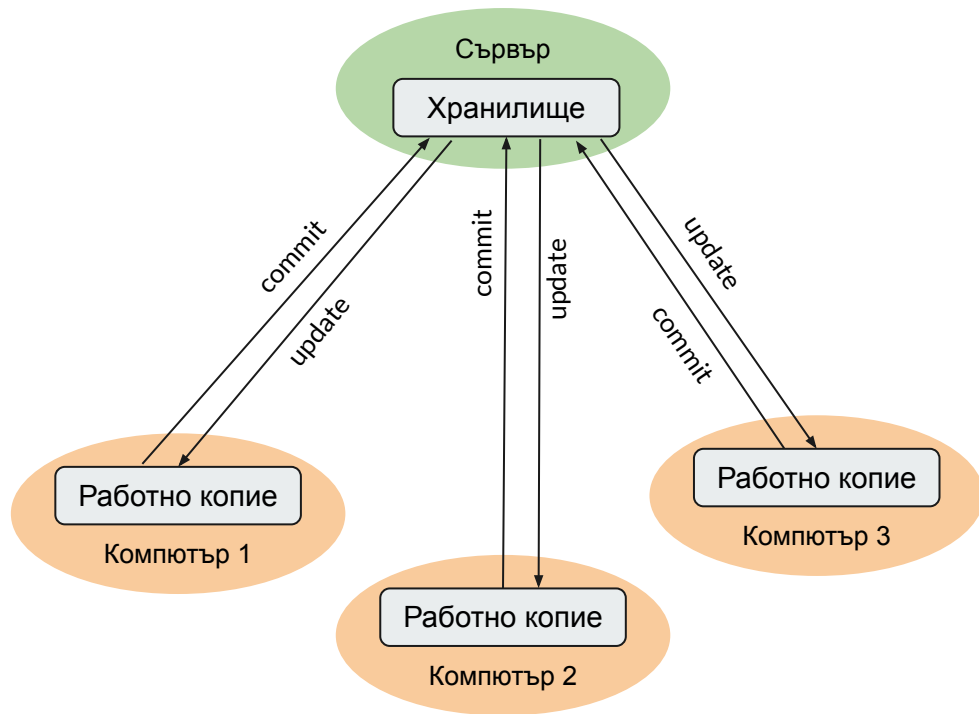
Разпределен и централизиран контрол на версиите

Съществуват два вида системи за контрол на версиите: централизирани (centralized) и разпределени (distributed).

- Основната разлика е в броя на хранилищата, които се използват - едно при централизирана система и множество при разпределена.

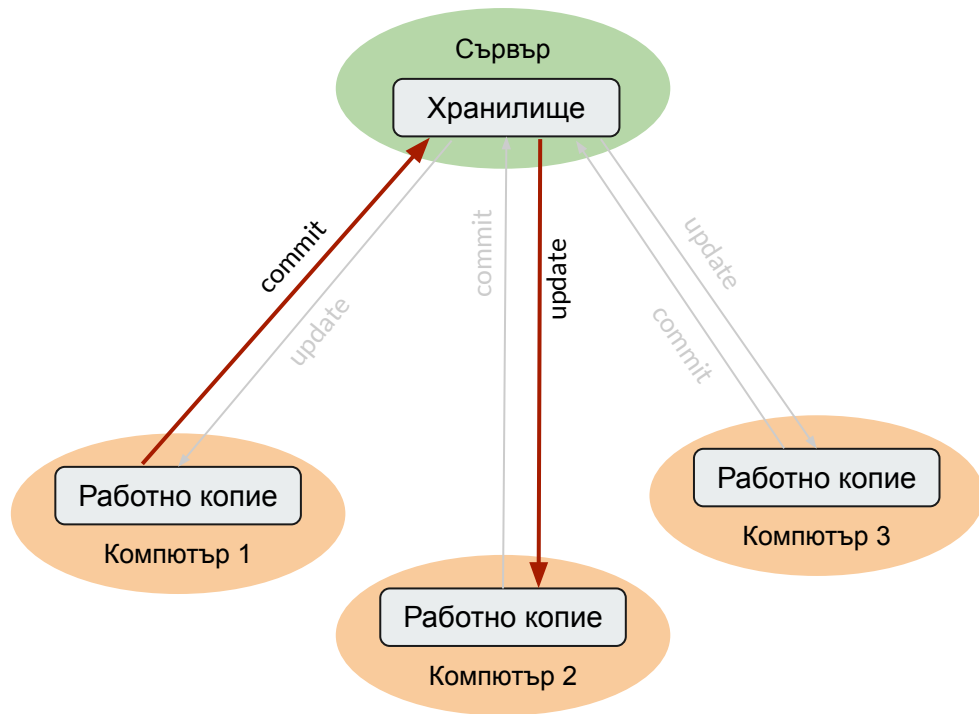
Централизирана система

- Всеки потребител работи със собствено копие, но има само едно централно хранилище (repository).
- Веднага след направен commit от един член на екипа останалите могат да направят update и да приложат промените върху собствените си локални копия.



Централизирана система

- Т.е. за да може някой да види вашите промени, трябва да се изпълнят следните стъпки:
 - Вие: commit
 - Той: update





Централизирана система

Ранните VCS са проектирани на базата на централизиран модел, при който всеки проект има само едно хранилище, използвано от всички разработчици. Този модел има два важни проблема.

- Единият е, че единственото хранилище е единична точка на срыв (single point of failure) - ако сървърът на хранилището не работи, цялата работа спира.
- Другият е, че трябва да сте свързани със сървъра, за да работите с промените; ако сте офлайн, не може да се работи.



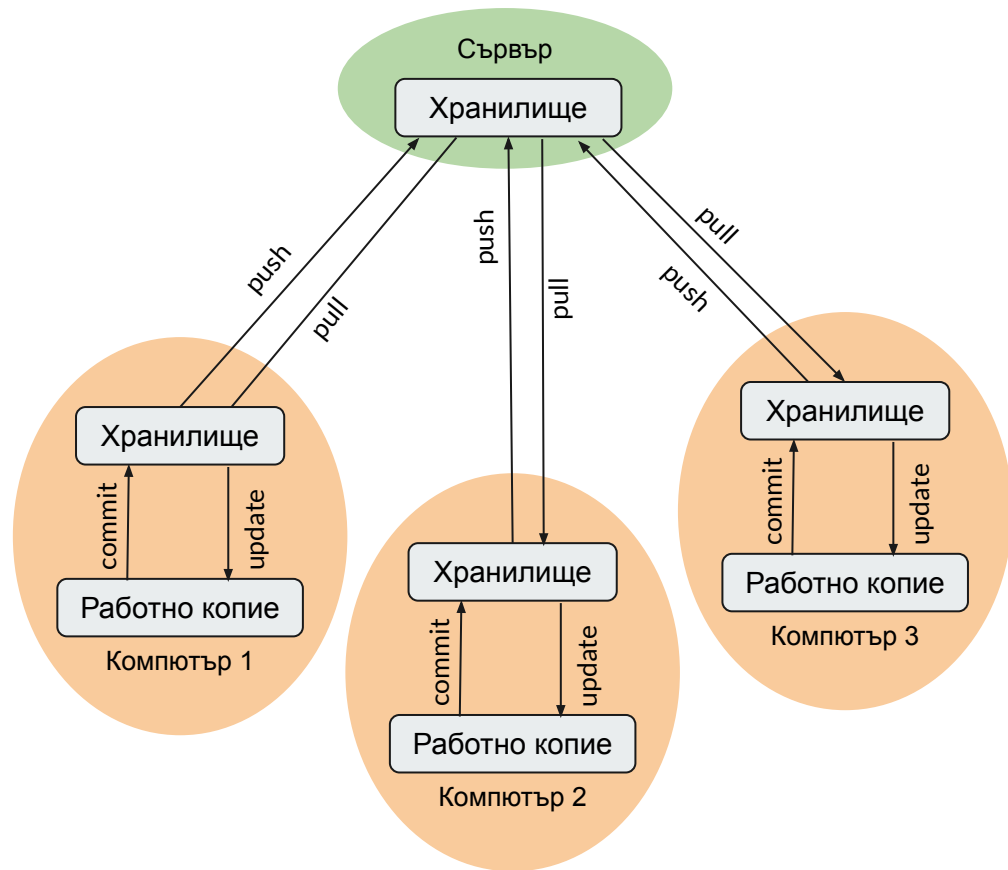
Централизирана система

Най-ранните VCS от първо поколение поддържат само локален достъп - т.е. всички разработчици на даден проект трябва да са на една и съща машина с единното централно хранилище на проекта.

VCS от второ поколение, макар и все още да централизират всеки проект около едно единствено хранилище, поддържат модела клиент-сървър, който позволява на разработчиците да работят с него по мрежата от други машини.

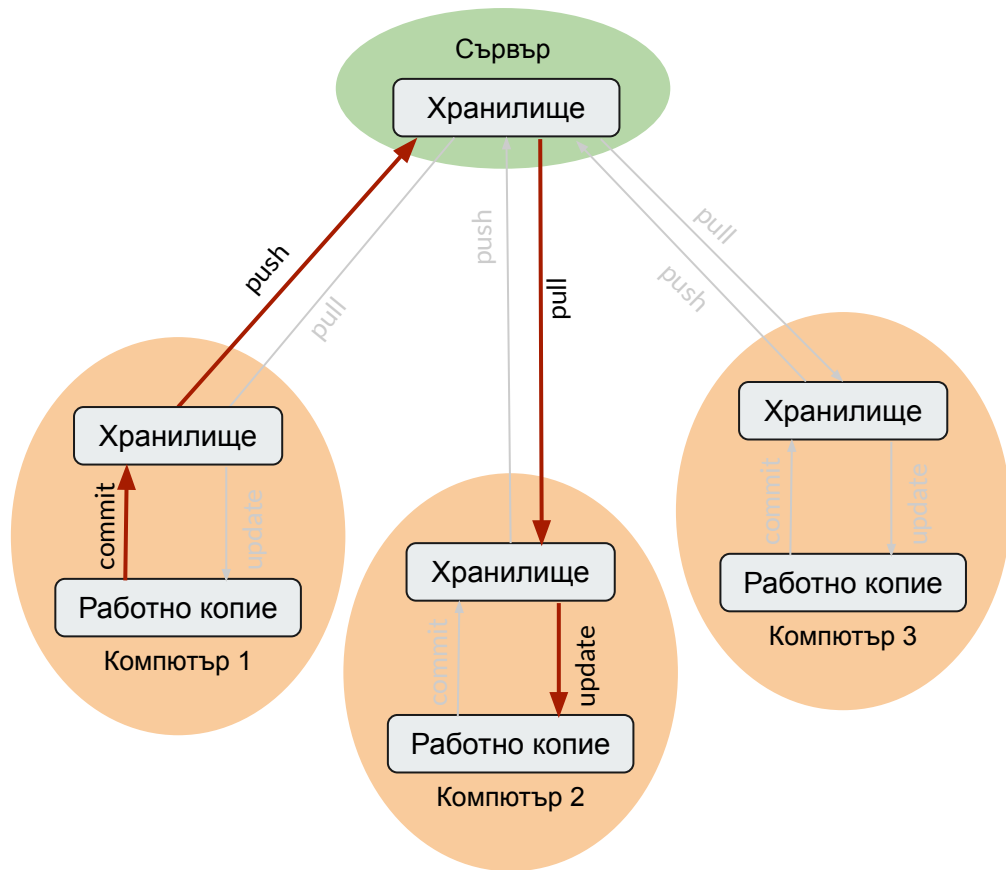
Разпределена система

- Всеки потребител има собствено *локално хранилище* и *работно копие*.
- След като даден потребител направи commit другите нямат достъп до неговите промени, защото те нямат достъп до локалното му хранилище.
- Необходимо е потребителя да направи **push** на промените от локалното си хранилище в централното.
- Когато потребител направи update той няма да получи промените от централното хранилище, а от локалното.
- За да се приложат промените от централното върху локалното хранилище трябва да се направи **pull**.



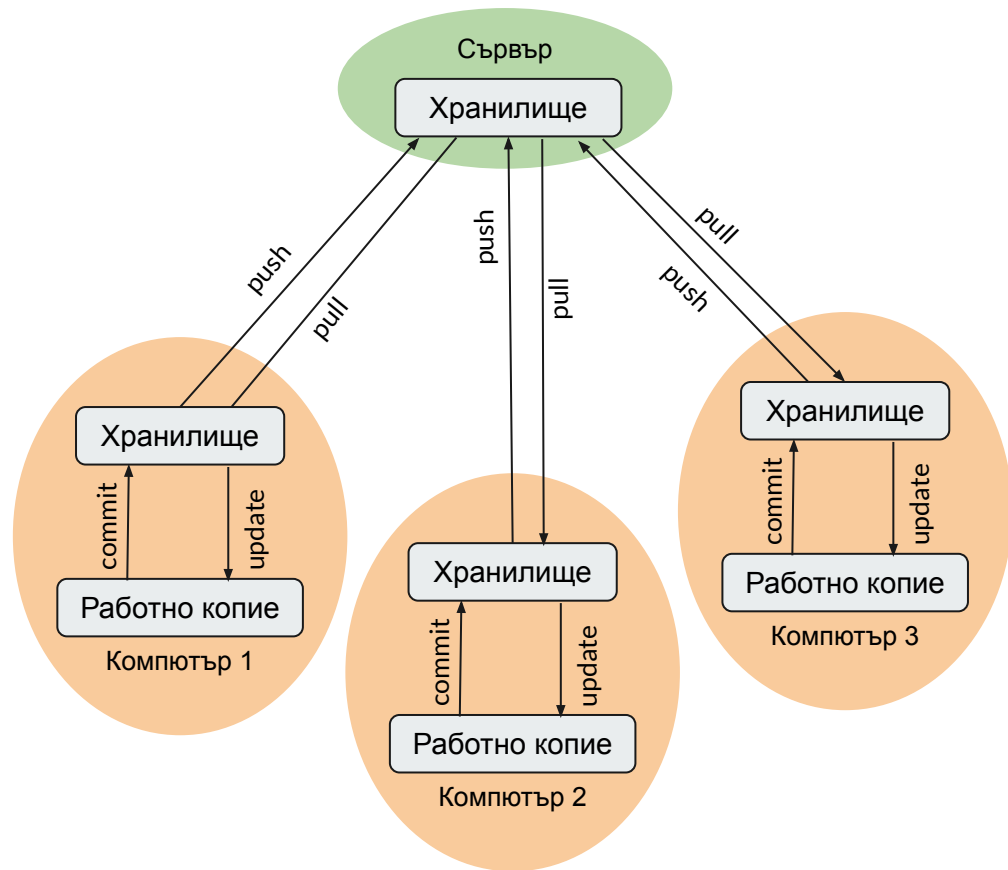
Разпределена система

- Т.е. за да може някой да види вашите промени, трябва да се изпълнят следните стъпки:
 - Вие: commit
 - Вие: push
 - Той: pull
 - Той: update



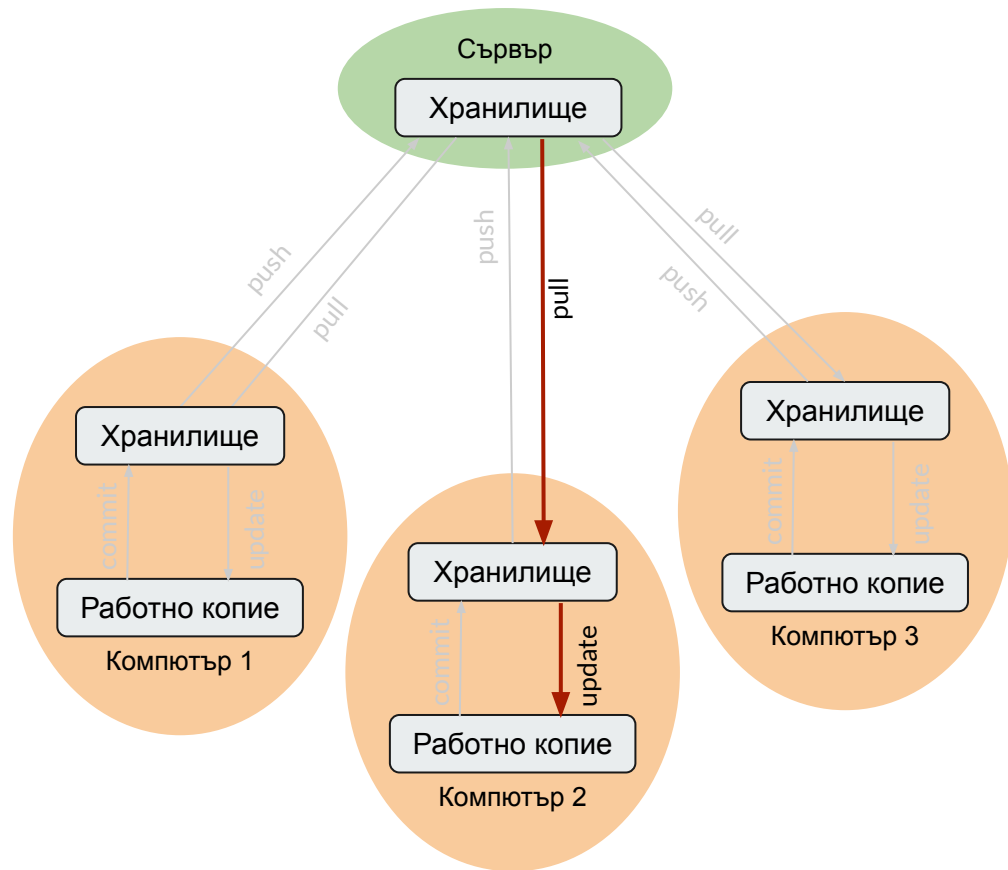
Разпределена система

- Командите **commit** и **update** преместват промените между *работното копие* и *локалното хранилище*, без да засягат *централното хранилище*.
- Командите **push** и **pull** преместват промените между *централното* и *локалното хранилища*, без да засягат *работното копие*.

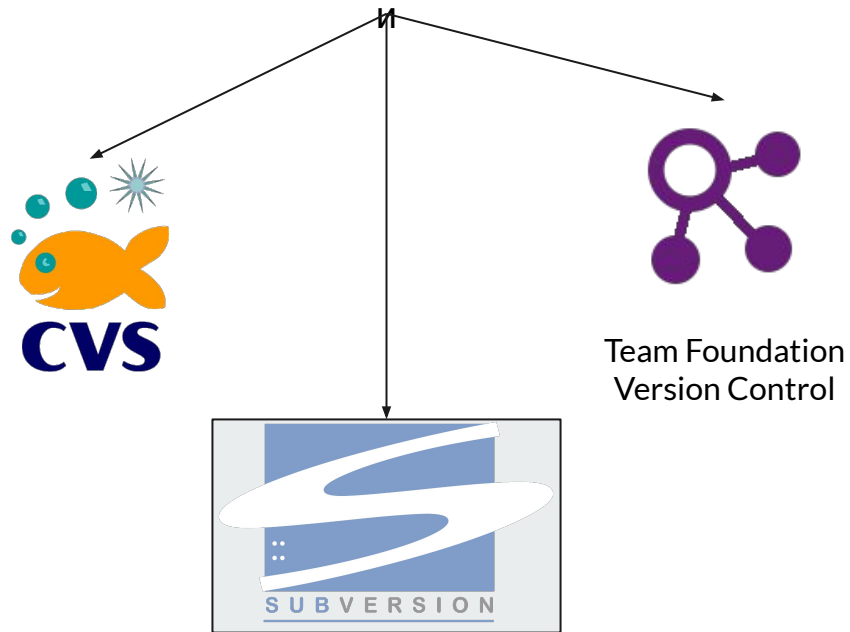


Разпределена система

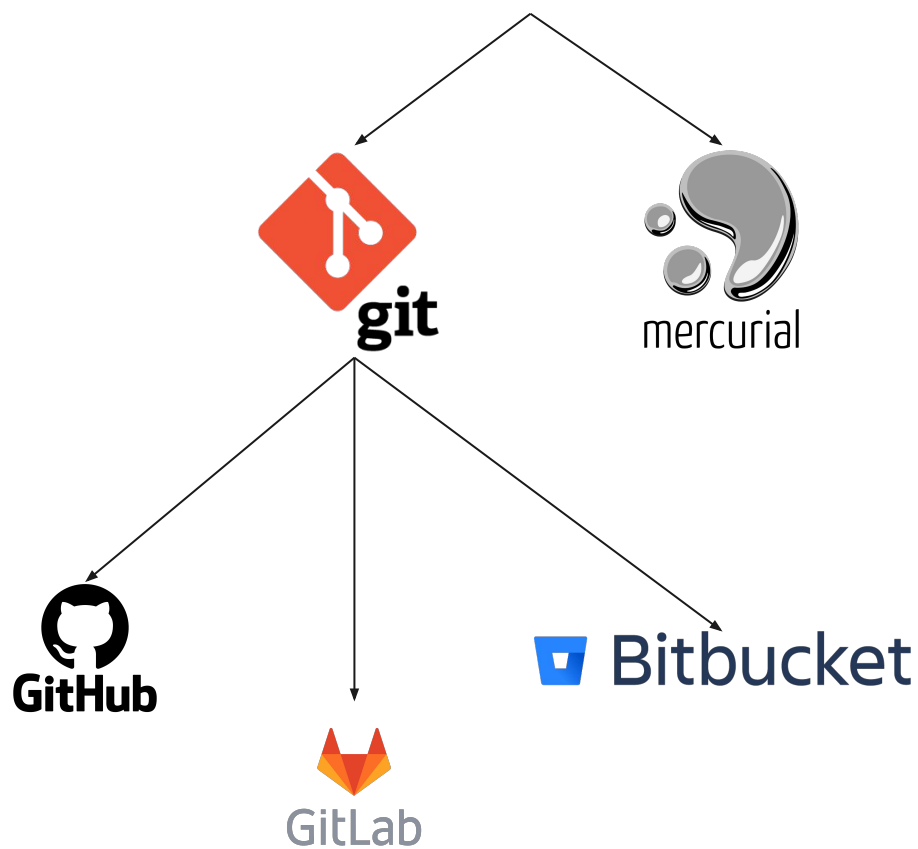
- Често е по-удобно командите **pull** и **update** да се изпълнят заедно последователно, за да се приложат последните промени от *централното хранилище* в *работното копие*.



Централизиран



Разпределени





<https://git.fmi.uni-plovdiv.bg>

Достъп с акаунт от [Е-портал](#)

ФМИ GitLab

DevOps платформа

Тази инстанция на GitLab се хоства във ФМИ на ПУ и е предназначена за подпомагане на обучението във факултета.

Ако сте студент използвайте акаунтът си в e-portal (<https://e-portal.uni-plovdiv.bg>) за вход в системата.

Преподавателите влизат с Focus акаунт.

E-portal Account

Standard

E-portal Account Username

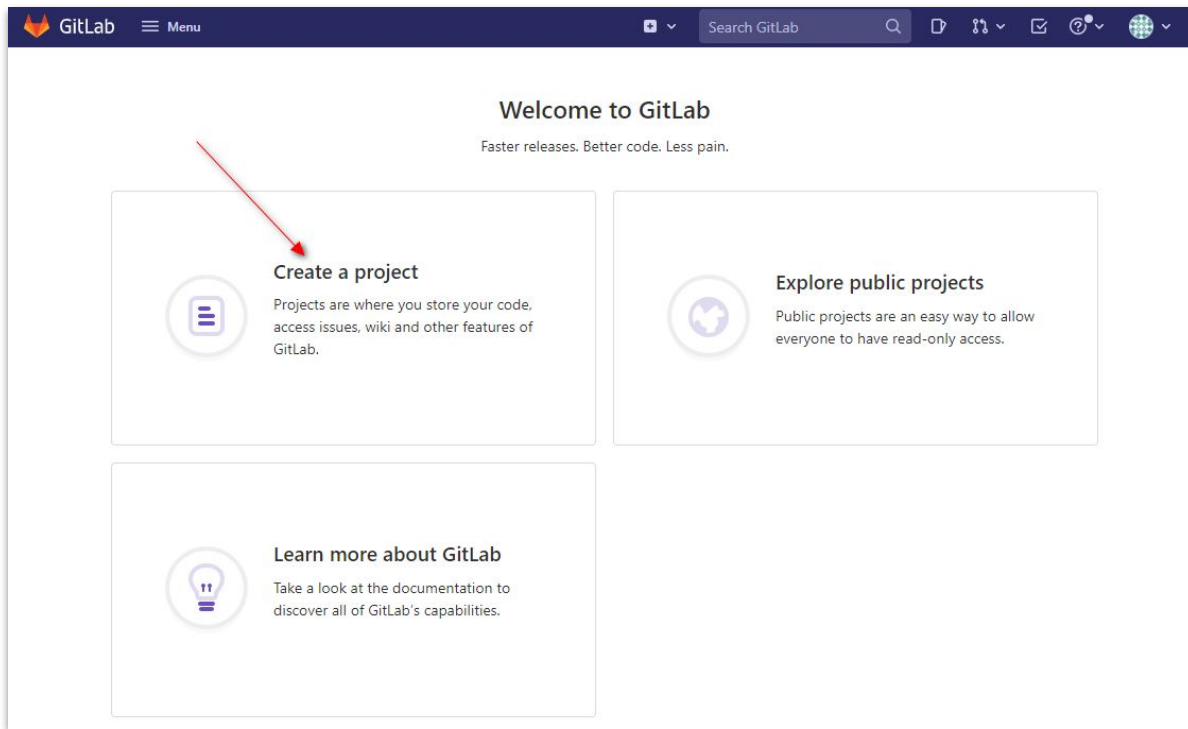
Password

☐ Remember me

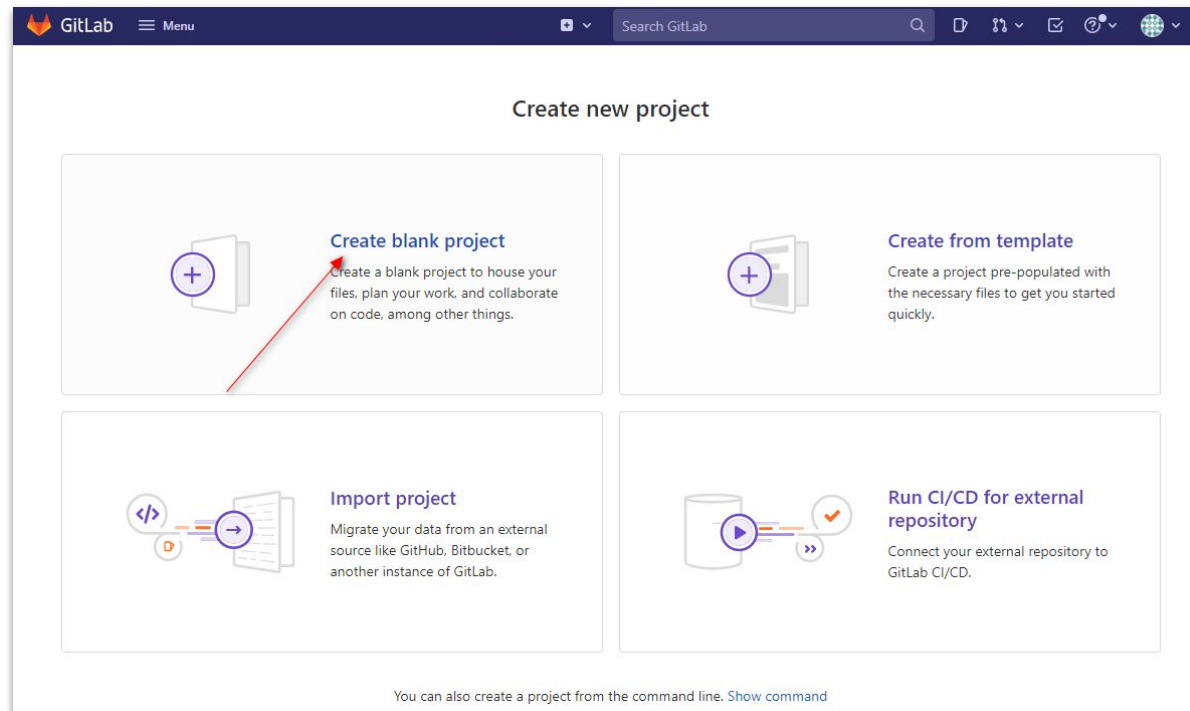
Sign in

Да не се бърка с официалния хостинг на GitLab: www.gitlab.com

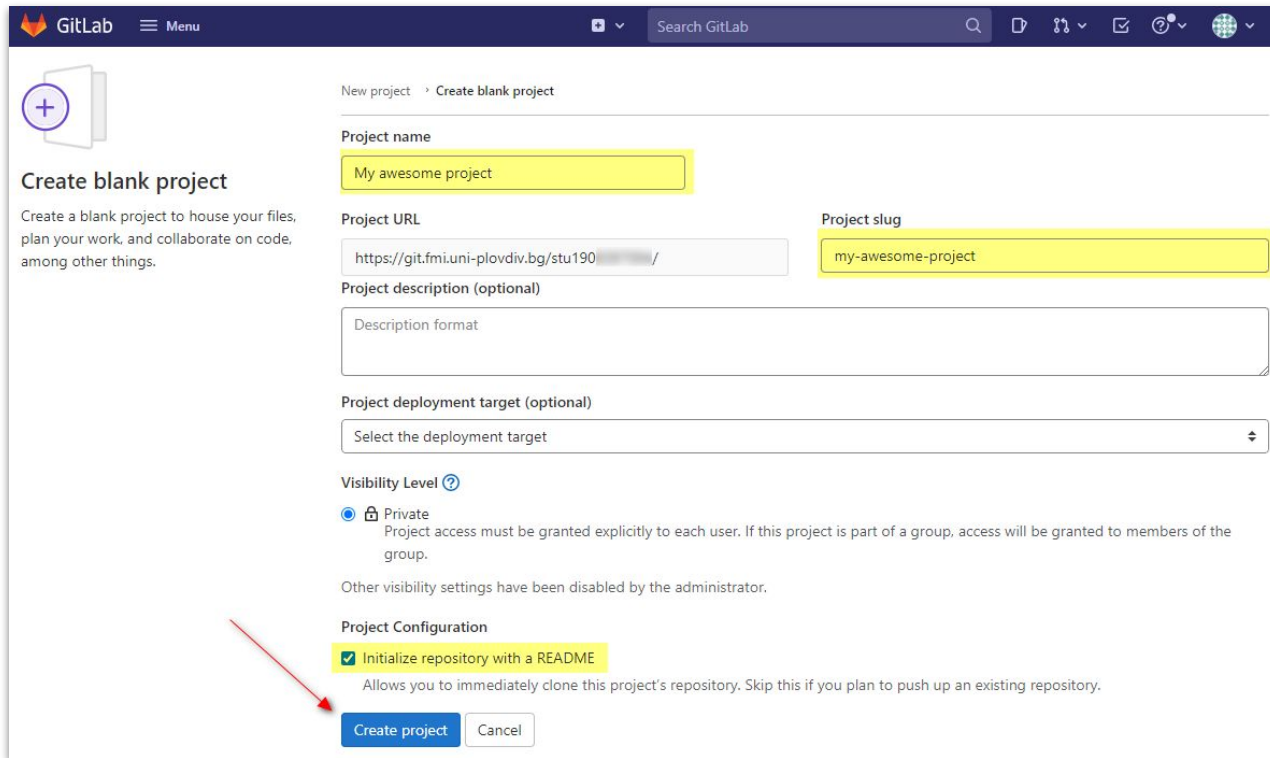
Стъпка 1: Създаване на проект в централното хранилище



Стъпка 1: Създаване на проект в централното хранилище



Стъпка 1: Създаване на проект в централното хранилище



The image shows the GitLab web interface for creating a new project. The header includes the GitLab logo, a menu icon, and a search bar. The main content area is titled 'New project' and 'Create blank project'. It contains several form fields: 'Project name' (filled with 'My awesome project'), 'Project URL' (filled with 'https://git.fmi.uni-plovdiv.bg/stu190/'), 'Project slug' (filled with 'my-awesome-project'), 'Project description (optional)' (empty), and 'Project deployment target (optional)' (a dropdown menu). Below these is the 'Visibility Level' section, which is set to 'Private'. At the bottom, the 'Project Configuration' section has a checked checkbox for 'Initialize repository with a README'. A red arrow points to the 'Create project' button at the bottom right.

GitLab Menu

New project Create blank project

Create blank project

Create a blank project to house your files, plan your work, and collaborate on code, among other things.

Project name

My awesome project

Project URL

https://git.fmi.uni-plovdiv.bg/stu190/

Project slug

my-awesome-project

Project description (optional)

Description format

Project deployment target (optional)

Select the deployment target

Visibility Level ?

☒ Private
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.

Other visibility settings have been disabled by the administrator.

Project Configuration

☒ Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project Cancel

Стъпка 1: Създаване на проект в централното хранилище

The screenshot displays the GitLab web interface for a newly created project named "My awesome project". The interface is divided into a left sidebar, a top navigation bar, and a main content area.


Left Sidebar: Contains a menu with the following items: "My awesome project" (selected), "Project information", "Repository", "Issues" (0), "Merge requests" (0), "CI/CD", "Security & Compliance", "Deployments", "Monitor", "Infrastructure", "Packages & Registries", "Analytics", "Wiki", "Snippets", and "Settings".

Top Navigation Bar: Includes the GitLab logo, a "Menu" button, a search bar labeled "Search GitLab", and several utility icons (notifications, settings, etc.).

Main Content Area:

- Project Header:** Shows the project name "My awesome project" with a lock icon, "Project ID: 927", and statistics: "1 Commit", "1 Branch", "0 Tags", "61 KB Files", and "61 KB Storage".
- Branches:** A dropdown menu shows "main" as the selected branch, followed by "my-awesome-project /" and a "+" button to add more branches.
- Initial Commit:** A section titled "Initial commit" shows a commit hash "432c30ab" and the text "authored just now".
- Quickstart Buttons:** A row of buttons for setting up the project: "Upload File", "README", "Add LICENSE", "Add CHANGELOG", "Add CONTRIBUTING", "Enable Auto DevOps", "Add Kubernetes cluster", "Set up CI/CD", and "Configure Integrations".
- Commit History Table:** A table with columns "Name", "Last commit", and "Last update". It contains one entry: "README.md" with "Initial commit" as the last commit and "just now" as the last update.
- README.md Content:** The content of the README file is displayed, starting with "My awesome project", followed by "Getting started" and "Add your files" sections.

Bottom of the Page: A "Collapse sidebar" button is visible in the bottom left corner.



Стъпка 1: Създаване на проект в централното хранилище

В централното хранилище е създаден нов проект, който съдържа един файл с име README.md.

Сървър

Хранилище

Локален компютър

Стъпка 2: Подготовка на локалното хранилище

За да се клонира състоянието на централното хранилище е необходимо да се вземе неговия URL, известен като remote URL.

The screenshot shows the GitLab web interface for a project named "My awesome project". The left sidebar contains a menu with options like Project information, Repository, Issues, Merge requests, CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings. The main content area displays the project details, including the project ID (927), commit count (1), branch count (1), tag count (0), file count (61 KB), and storage count (61 KB). A red arrow points to the "Clone" button in the top right corner. Below the clone button, there is a section for the initial commit, showing the commit hash (432c30ab) and the commit message (Initial commit). The "README.md" file is listed in the file browser, showing the initial commit. The "README.md" content is displayed below, starting with "My awesome project" and "Getting started".

GitLab Menu

Search GitLab

My awesome project

My awesome project

Project ID: 927

1 Commit 1 Branch 0 Tags 61 KB Files 61 KB Storage

main my-awesome-project / +

History Find file Web IDE Clone

Initial commit

432c30ab

Upload File README Add LICENSE Add CHANGELOG Add CONTRIBUTING Enable Auto DevOps

Add Kubernetes cluster Set up CI/CD Configure Integrations

Name	Last commit	Last update
README.md	Initial commit	just now

README.md

My awesome project

Getting started

To make it easy for you to get started with GitLab, here's a list of recommended next steps.

Already a pro? Just edit this README.md and make it your own. Want to make it easy? [Use the template at the bottom!](#)

Add your files

☐ Create or upload files

☐ Add files using the command line or push an existing Git repository with the following command:

Стъпка 2: Подготовка на локалното хранилище

За да се клонира състоянието на централното хранилище е необходимо да се вземе неговия URL, известен като remote URL.

The screenshot shows the GitLab web interface for a project named "My awesome project". The left sidebar contains a menu with options: Project information, Repository, Issues (0), Merge requests (0), CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings. The main content area displays the project details, including the project ID (927), 1 commit, 1 branch, 0 tags, 61 KB files, and 61 KB storage. A dropdown menu for cloning is open, showing the "Clone with HTTPS" option with the URL `https://git.fmi.uni-plovdiv.bg/stu` and a red arrow pointing to the "Clone" button. Below the cloning options, there is a table with the following data:

Name	Last commit	Last update
README.md	Initial commit	just now

Below the table, the "README.md" file is displayed, showing the project name "My awesome project" and the "Getting started" section. The "Getting started" section includes a list of recommended next steps and a link to the template at the bottom. The "Add your files" section includes a list of options: "Create or upload files" and "Add files using the command line or push an existing Git repository with the following command:".

Стъпка 2: Подготовка на локалното хранилище

На локалния компютър се изпълнява командата за клониране на централно хранилище в локално:

```
git clone [remote URL]
```

C:\Windows\system32\cmd.exe

```
D:\Temp>git clone https://git.fmi.uni-plovdiv.bg/stu190[redacted]/my-awesome-project.git
Cloning into 'my-awesome-project'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

D:\Temp>
```



Сървър

Хранилище

Стъпка 2: Подготовка на локалното хранилище

По този начин *локалното хранилище* (local repository) е създадено и е свързано с *централното* (remote repository / origin). Освен това е инициализирано и *работното копие*.

Директорията `.git` е системна и е необходима за контрола на версиите. Тя не бива да се изтрива.

HDD (D:) > Temp > my-awesome-project				
Име	Дата на промяна	Тип	Размер	
 .git	4.4.2022 г. 1:07	Папка с файлове		
 README.md	4.4.2022 г. 1:07	MD файл	7 KB	

Хранилище

Работно копие

Локален компютър



Стъпка 3: Операции и команди

Добавяне на нов файл към контрола на версиите:

```
git add Application.java
```



Стъпка 3: Операции и команди

Проверка на състоянието на *работното копие*:

```
git status
```

```
D:\Temp\my-awesome-project>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   Application.java
```


Сървър

Хранилище

Стъпка 3: Операции и команди

Commit на промените от *работното копие* в *локалното хранилище*:

```
git commit -m "Описание на промените"
```

```
D:\Temp\my-awesome-project>git commit -m "My first file"  
[main d4ac3fd] My first file  
1 file changed, 2 insertions(+)  
create mode 100644 Application.java
```

Хранилище

commit
↑

Работно копие

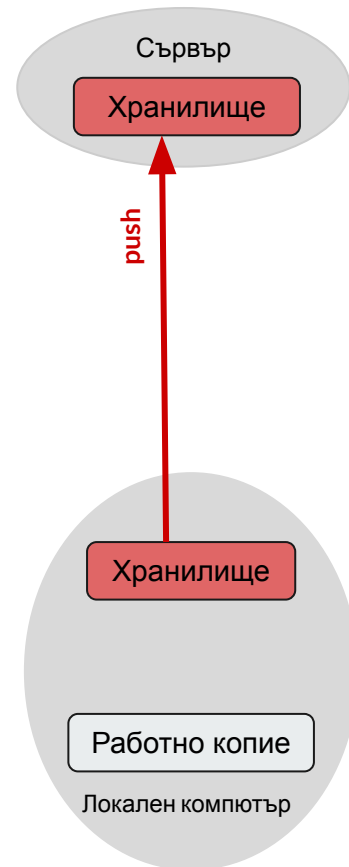
Локален компютър

Стъпка 3: Операции и команди

Push на промените от *локалното хранилище* към *централното*:

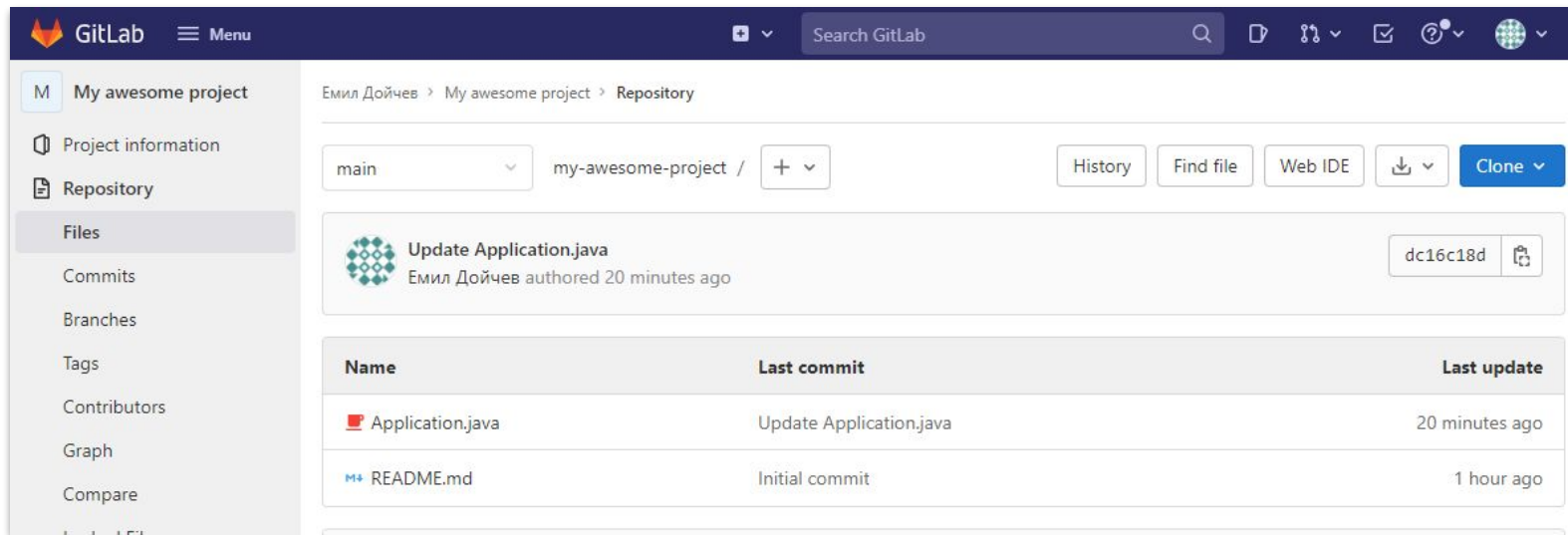
```
git push
```

```
D:\Temp\my-awesome-project>git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 316 bytes | 316.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://git.fmi.uni-plovdiv.bg/stu190 /my-awesome-project.git
432c30a..d4ac3fd main -> main
```



Стъпка 3: Операции и команди

Състояние на *централното хранилище*:



The screenshot displays the GitLab web interface for a repository named "My awesome project". The left sidebar contains navigation links: "Project information", "Repository", "Files", "Commits", "Branches", "Tags", "Contributors", "Graph", and "Compare". The main content area shows the repository's state, including a dropdown for the "main" branch, the repository path "my-awesome-project /", and buttons for "History", "Find file", "Web IDE", "Clone", and "Download". A commit titled "Update Application.java" by "Емил Дойчев" is highlighted, showing a commit hash of "dc16c18d". Below this, a table lists the repository's files and their commit history.

Name	Last commit	Last update
Application.java	Update Application.java	20 minutes ago
README.md	Initial commit	1 hour ago

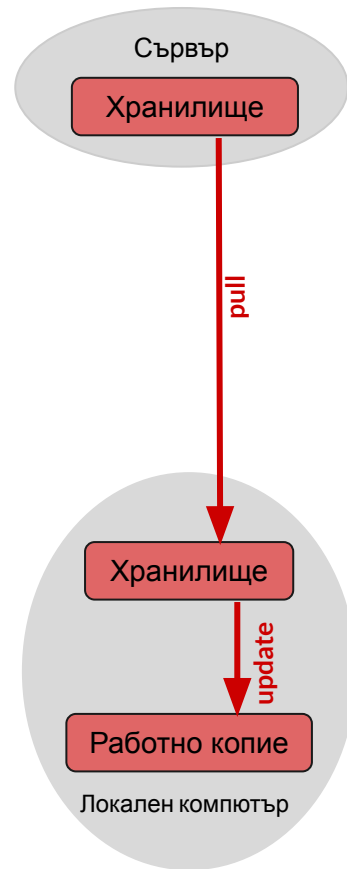
Стъпка 3: Операции и команди

Pull на промените от *централното хранилище* в *локалното*:

```
git pull
```

```
D:\Temp\my-awesome-project>git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://git.fmi.uni-plovdiv.bg/stu190 /my-awesome-project
d4ac3fd..dc16c18 main -> origin/main
Updating d4ac3fd..dc16c18
Fast-forward
 Application.java | 5 ++++-
 1 file changed, 4 insertions(+), 1 deletion(-)
```

Командата **pull** на git реално извършва операциите **pull** и **update** заедно и последователно. Т.е. тя **не е симетрична** на операцията **push**.





Стъпка 3: Операции и команди

Списък на промените (commits) в локалното хранилище:

```
git log
```

```
$ git log
commit dc16c18d0ddb1e1c42066e4cc1d4b91762779fd3 (HEAD -> main, origin/main, origin/HEAD)
Author: Емил Дойчев <stu190@uni-plovdiv.bg>
Date: Sun Apr 3 22:38:15 2022 +0000

    Update Application.java

commit d4ac3fd5c5b8082fc7072ca94d80f98769ca3476
Author: Emil Doychev <e.doychev@uni-plovdiv.bg>
Date: Mon Apr 4 01:29:58 2022 +0300

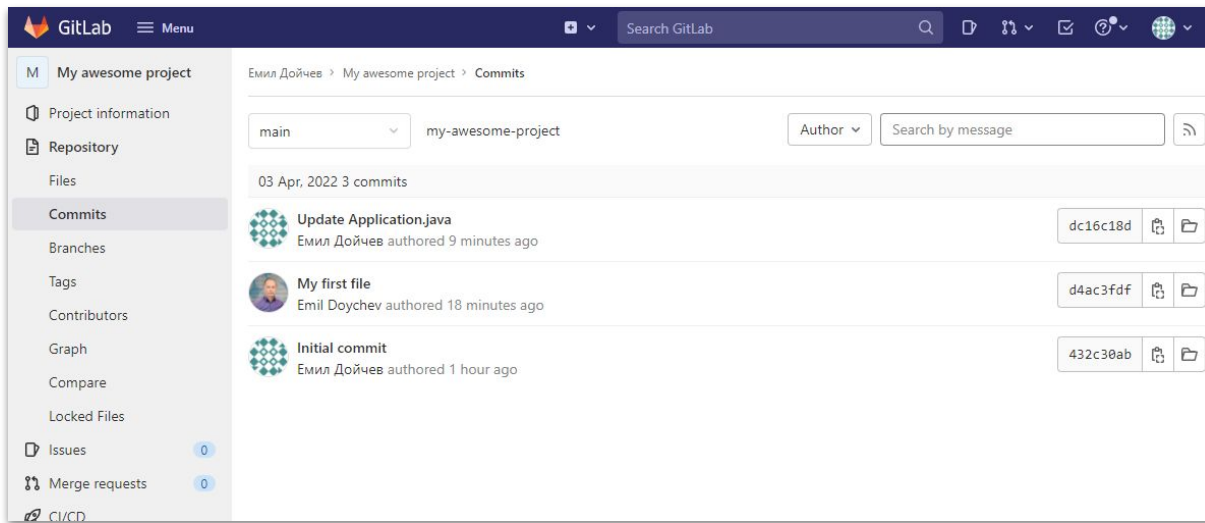
    My first file

commit 432c30abc0d4eb66504d0c68afc702f34a9d6d30
Author: Емил Дойчев <stu190@uni-plovdiv.bg>
Date: Sun Apr 3 21:47:51 2022 +0000

    Initial commit
```

Стъпка 3: Операции и команди

Списък на промените (commits) в *централното хранилище*:



The screenshot displays the GitLab web interface for a project named "My awesome project". The left sidebar contains navigation links: "Project information", "Repository", "Files", "Commits" (highlighted), "Branches", "Tags", "Contributors", "Graph", "Compare", "Locked Files", "Issues" (0), "Merge requests" (0), and "CI/CD". The main content area shows the commit history for the "main" branch of "my-awesome-project". It lists three commits by "Емил Дойчев" (Emil Doychev):

- Update Application.java** (dc16c18d) - authored 9 minutes ago
- My first file** (d4ac3fdf) - authored 18 minutes ago
- Initial commit** (432c30ab) - authored 1 hour ago

Each commit entry includes a commit icon, the commit message, the author's name, the time since authoring, the commit hash, and icons for cloning and viewing the commit details.



Разрешаване на конфликти

Системата за контрол на версиите трябва да има някакъв механизъм за предотвратяване или разрешаване на конфликти между потребители, които искат да променят един и същ файл.

Съществуват три различни начина за постигане на тази цел:

- Заклучване (locking)
- Обединяване преди прилагане на промените (merge before commit)
- Прилагане на промените преди обединяване (commit before merge)



Конфликти: Locking

При заключване работните файлове обикновено са само за четене, така че не могат да бъдат променяни.

Ако е необходимо да се извърши промяна трябва да се изиска от VCS да направи даден работен файл достъпен за писане, като го заключи за останалите потребители. Само един потребител може да заключи файл в даден момент.

Когато промените бъдат приложени (commit) това отключва файла и той отново става само за четене. Това позволява на други потребители да заключат файла, за да направят допълнителни промени.



Конфликти: Locking

Когато стратегията за заключване работи добре, работният процес изглежда по следния начин:

1. Ваня заключва файла `foo.java` и започва да го променя.
2. Боби, който се опитва да модифицира `foo.java`, е уведомен, че Ваня го е заключила и той не може да го изтегли за промяна.
3. Боби е блокиран и не може да продължи. Затова убива времето с чаша кафе.
4. Ваня завършва промените си и ги предава (`commit`), което отключва `foo.java`.
5. Боби допива кафето си, връща се и отваря `foo.java` за редакция, като го заключва за достъп от други потребители.



Конфликти: Locking

За съжаление работният процес твърде често изглежда по-скоро така:

1. Ваня заключва файла `foo.java` и започва да го променя.
2. Боби, който се опитва да модифицира `foo.java`, е уведомен, че Ваня го е заключила и той не може да го редактира.
3. Междувременно Ваня получава напомняне, че закъснява за среща, и бърза да отиде на нея, като оставя `foo.java` заключен.
4. Боби, който отново се опитва да модифицира `foo.java`, отново е уведомен, че Ваня го е заключила и той не може да го модифицира.
5. След като два пъти е бил възпрепятстван и е загубил значителна част от деня си, чакайки отключването на `foo.java`, Боби проклина Ваня с чувство. Той уведомява VCS, че иска да открадне ключалката.
6. Ваня се връща от среща и намира email или мигновено съобщение, което я информира, че Боби е откраднал ключалката ѝ.
7. Промените в работното копие на Ваня сега са в конфликт с тези на Боби и ще трябва да бъдат обединени по-късно. Заклучването се оказва безполезно.



Конфликти: Locking

В исторически план заключването е първият метод за разрешаване на конфликти и се свързва с първото поколение централизирани VCS, поддържащи само локален достъп.



Конфликти: Обединяване преди прилагане на промените (merge before commit)

В система за обединяване преди предаване VCS забелязва, когато се прави опит за предаване на файл или файлове, които са се променили междувременно. Тогава VCS изисква първо да бъде разрешен конфликта и след това да се извърши предаването.



Конфликти: Обединяване преди прилагане на промените (merge before commit)

Работният процес обикновено изглежда по следния начин:

- Ваня изтегля копие на файла `foo.java` и започва да го променя.
- Боби също изтегля копие на файла `foo.java` и започва да го променя.
- Ваня завършва промените си и ги предава.
- Боби, който се опитва да предаде файла, е информиран, че версията в хранилището се е променила след неговото изтегляне и той трябва да разреши конфликта, преди да извърши предаването.
- Боби изпълнява команда за обединяване, която прилага промените на Ваня към неговото работно копие.
- Промените на Ваня и Боби във `foo.java` всъщност не се припокриват. Командата за обединяване минава успешно и VCS позволява на Боби да предаде обединената версия.



Конфликти: Обединяване преди прилагане на промените (merge before commit)

Исторически погледнато, обединяването преди предаване е вторият възникнал метод за разрешаване на конфликти и се свързва с централизирани VCS от типа клиент-сървър.



Конфликти: Предаване преди обединяване (commit before merge)

Възможно е VCS никога да не блокира предаването. Вместо това, ако копието на хранилището се е променило, след като файловете са били изтеглени за модификация, предаването може просто да бъде изпълнено в нов клон (branch).

В последствие клоновете могат да останат отделни или всеки разработчик може да извърши обединяване, което да ги събере отново.

Моделът "commit-before-merge" се свързва с децентрализираните VCS от трето поколение и се появява около 1998 г.



Важни концепции, които пропускаме сега

- Разрешаване на конфликти (resolve conflicts)
- Обединяване на промени (merge)
- Организиране на промени в commit
 - Логическа свързаност на промените в един commit
 - Описание на commit
- Изключване на автоматично генерираните файлове от контрола на версиите

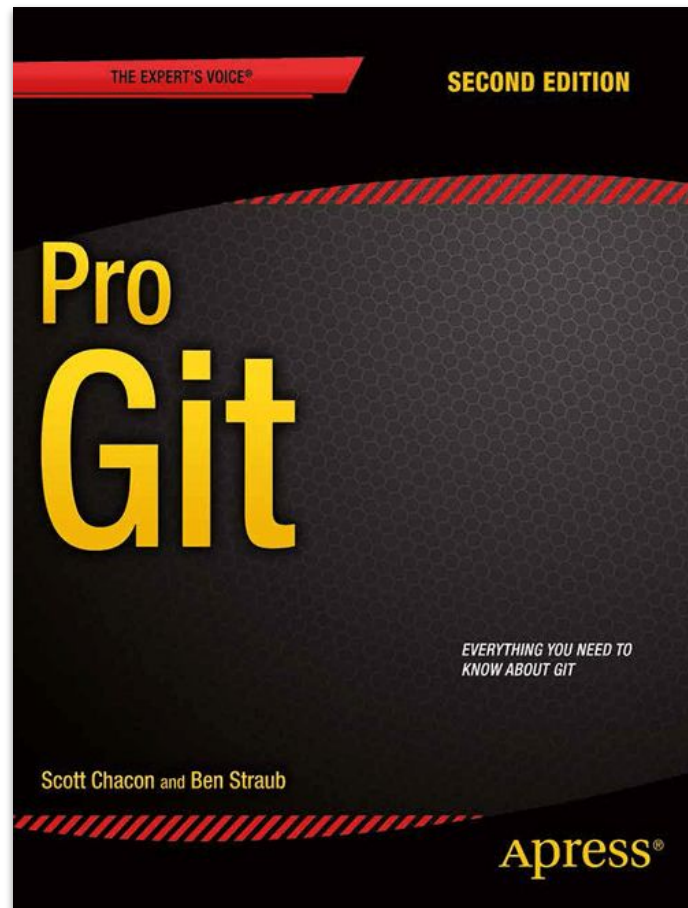


Литература

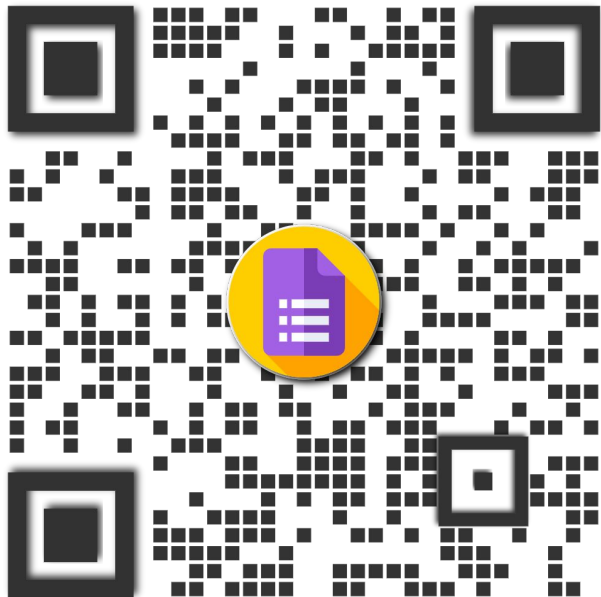
<https://git-scm.com/book/bg/v2>

На български език

Свободно достъпна



Регистриране на присъствие



https://t.ly/MY_Y

Отговор: SVN