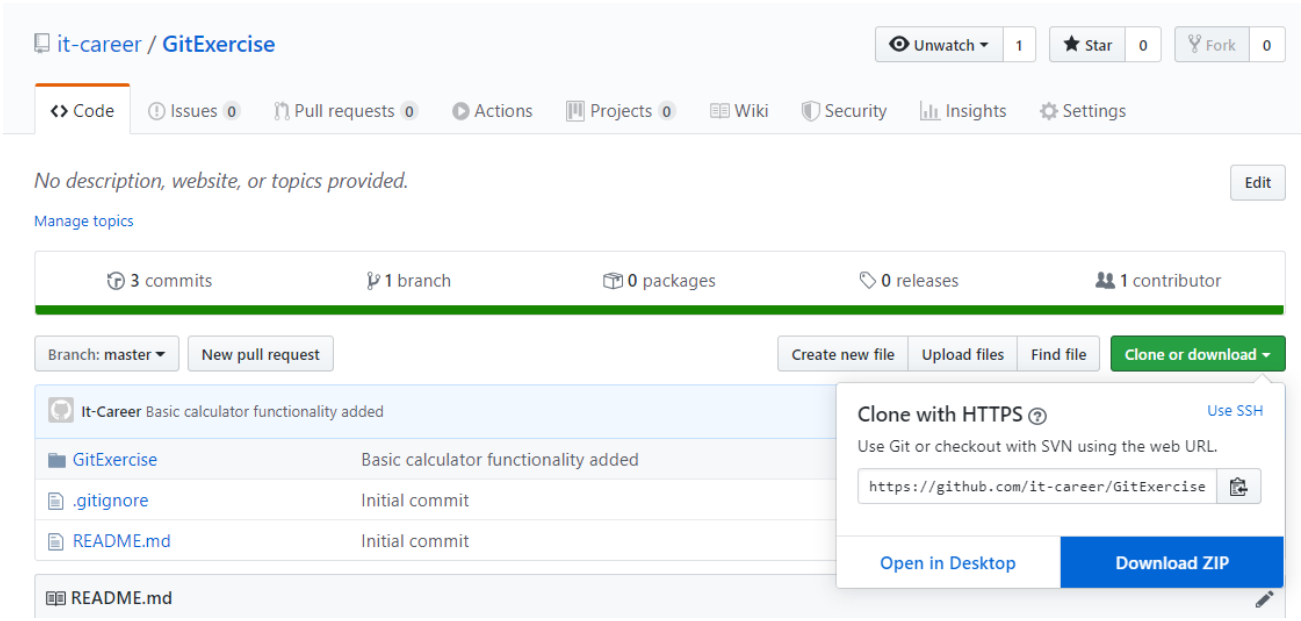
Упражнение: Git в практиката

1. Какво ще постигнем

Следвайки упражнението, ще упражните основните Git команди, които всеки разработчик на софтуер използва ежедневно. За целта се разделете на екипи от по трима души. Всеки от екипа ще има собствена задача, която ще изисква от него да използва научените Git команди. За улеснение ще наречем хората в екипа Разработчик А, Разработчик Б и Разработчик В. За упражнението ще използваме малък и много опростен проект – конзолен калкулатор. По този начин ще се фокусираме върху самите Git команди, а не върху имплементацията на сложни алгоритми.

2. Хост на репозитори

Проектът е хостнат в GitHub и може да бъде открит на адрес https://github.com/it-career/GitExercise. За да започнете работата си по него, нека вътрешно избран член на екипа да го изтегли на машината си.



След като проектът е изтеглен, трябва да се създаде локално Git репозитори, като преди това е добре да се конфигурират личните данни за потребителят (тази стъпка е за всички членове на екипа).

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop
$ git config --global user.name "IT-Career"

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop
$ git config --global user.email "youremail@mail.com"
```

След като всеки потребител конфигурира личните си данни, е време да пристъпим към създаването на локално репозитори.

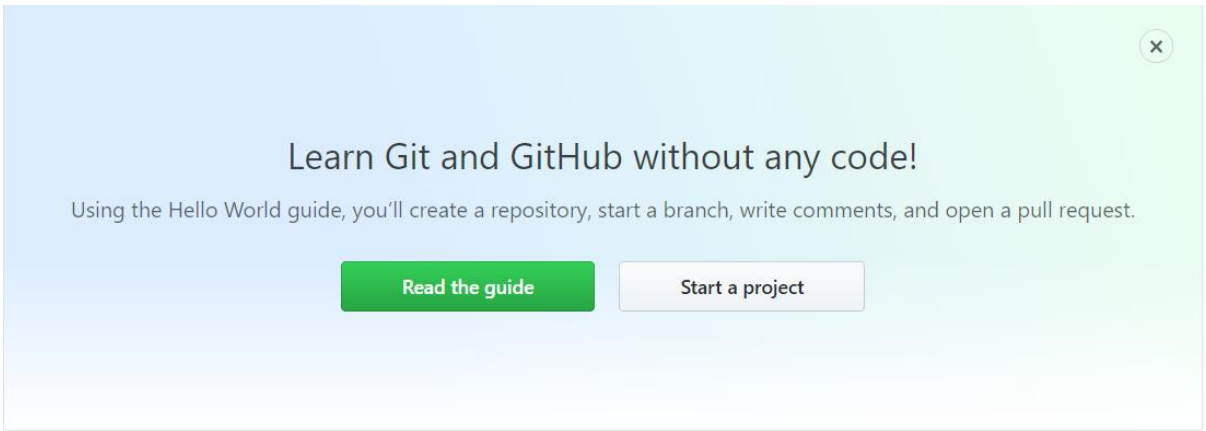
Тази стъпка трябва да бъде изпълнена от члена на екипа, който е изтеглил проекта. Първо, използвйки конзолата, трябва да отидем в директорията на изтегления проект (използвайте командата cd <directory_path>, а за да сте сигурни, че сте на правилното място можете да проверите съдържанието на папката с командата ls).

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop
$ cd C:/Users/danai/Desktop/GitExerciseV2

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/GitExerciseV2
$ ls
GitExercise/  README.md
```

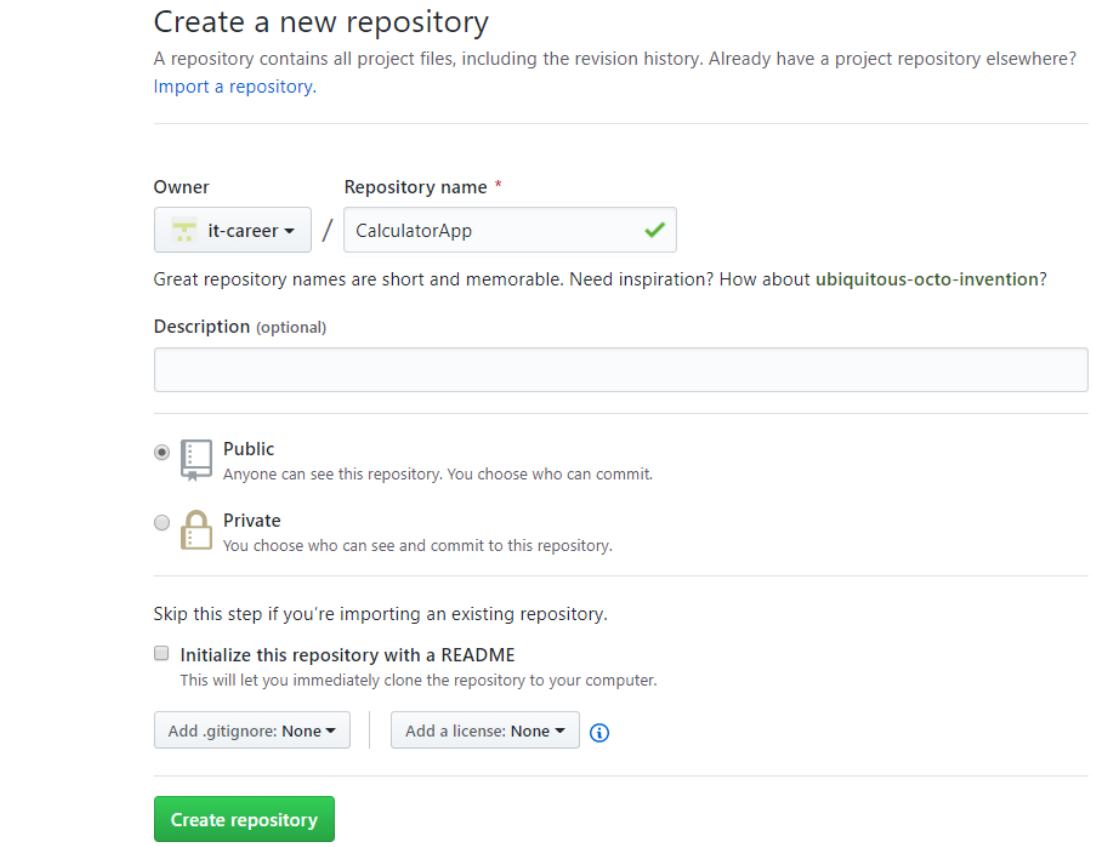
След като сме сигурни, че сме в правилната директория е време да инициализираме локално репозитори за нашия проект.

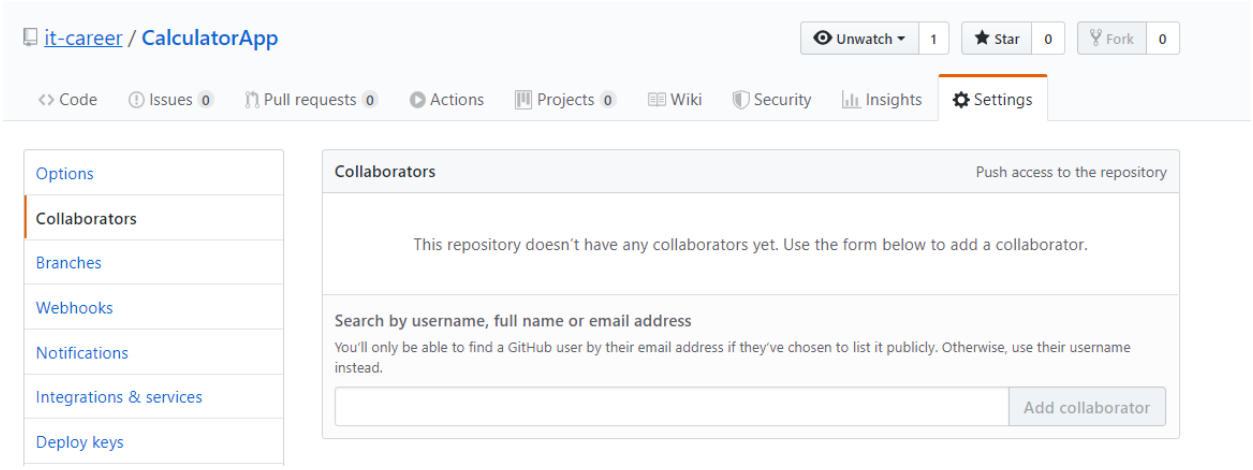
```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/GitExerciseV2
$ git init
Initialized empty Git repository in C:/Users/danai/Desktop/GitExerciseV2/.git/
```

Време е да хостнем репозиторито си в GitHub. От https://github.com/, след като влезете в акаунта си, старитрайте нов проект.



Дайте му подходящо име и изберете дали да е публичен или частен. В този случай ние ще създадем публичен проект. След създаването на проекта трябва да се добавят другите членове на екипа към проекта, за да могат да работят по него директно в GitHub хостнатото репозитори. За целта отидете на Options -> Collaborators и въведете потребителското име на вашите колеги.





След като сме изпълнили и тези стъпки е време да хостнем локалния си проект на току-що създаденото от нас GitHub репозитори.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/GitExerciseV2 (master)
$ git remote add origin https://github.com/it-career/CalculatorApp.git
```

Нека проверим текущия статус на нашето репозитори.

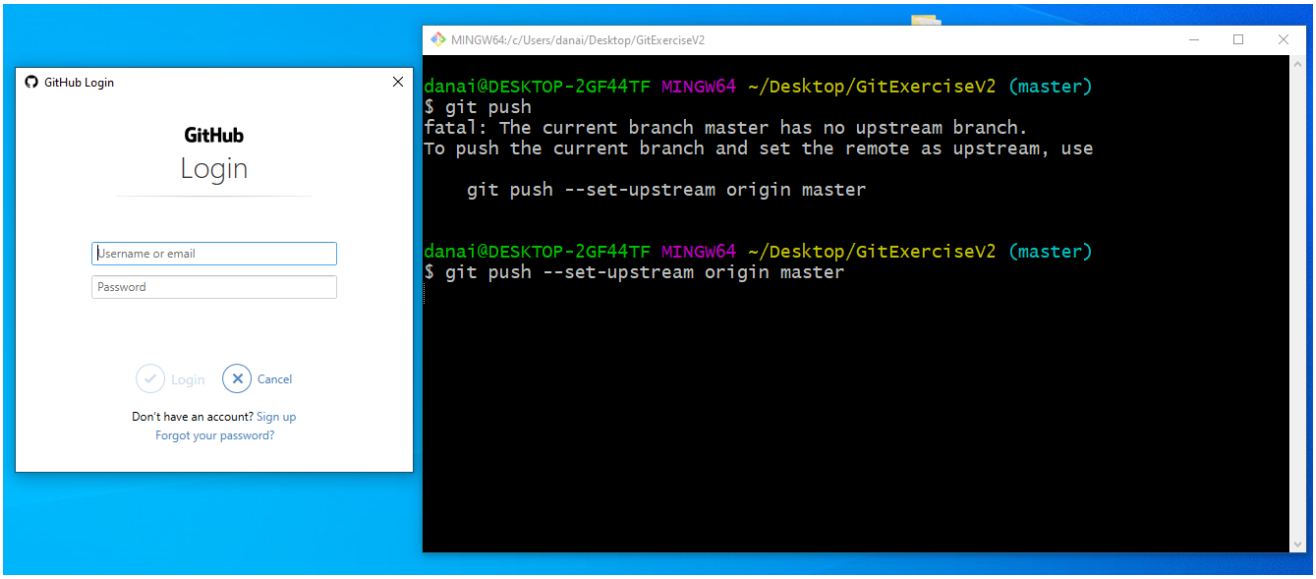
```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/GitExerciseV2 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        GitExercise/
        README.md

nothing added to commit but untracked files present (use "git add" to track)
```

След като се уверим, че всичко изглежда наред е време да добавим промените и да ги commit-нем към GitHub, за да качим началния вид на проекта. Ще използваме командата git add . , за да добавим за следене файловете, намиращи се в нашата директория, след което ще ги пакетираме в commit, за който ще изберем подходящо съобщение.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/GitExerciseV2 (master)
$ git add .
```

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/GitExerciseV2 (master)
$ git commit -m "Project upload"
[master (root-commit) 8871968] Project upload
 6 files changed, 436 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 GitExercise/GitExercise.csproj
 create mode 100644 GitExercise/GitExercise.sln
 create mode 100644 GitExercise/OptionsManager.cs
 create mode 100644 GitExercise/Startup.cs
 create mode 100644 README.md
```

Възможно е, при опит да изпратим commit-а към сървъра (git push) да получим грешка, защото за текущия клон няма зададено указание, къде да изпраща промените. Git е достатъчно умен и ще ни даде и решението на този проблем. След тази настройка ще ни бъдат поискани данните за нашия GitHub акаунт. Това се случва еднократно, след което тези данни се запазват и няма да ни бъдат поискани пак. Това е начинът, по който GitHub определя дали даден потребител има права да прави промени по даден проект. Инструкции как може да стане това има и на страницата на новосъздаденото от вас GitHub репозитори.




```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/GitExerciseV2 (master)
$ git push -u origin master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 4.14 KiB | 1.03 MiB/s, done.
Total 9 (delta 0), reused 0 (delta 0)
To https://github.com/it-career/CalculatorApp.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

На този етап проектът е хостнат в GitHub и всеки член на екипа може да го свали, за да започне работа по него. Вътрешно в екипа изберете кой ще заеме ролята на разработчик А, Б и В.

3. Клониране

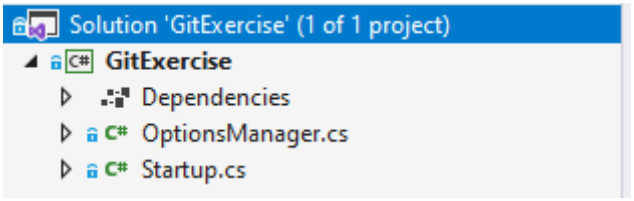
Всеки член на екипа трябва да има проекта в текущата си версия на своята машина. За да стане това, нека членовете на екипа го клонират, използвайки командата git clone <URL> - с HTTPS адреса на репозиторито, в избрана от потребителя директория.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop
$ git clone https://github.com/it-career/CalculatorApp.git
Cloning into 'CalculatorApp'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
```

След като всеки има проекта на своята машина, е време да се запознаем с проекта, който сме клонирали.

4. Проектът

Отворете проекта чрез Visual Studio. От Solution Explorer-а може да забележите, че проекта съдържа само един слой – конзолно приложение с класове в него, като единият съдържа Main метода, а другият е статичен клас, който изпълнява помощна функция. Нека всеки отдели време да се запознае с проекта, за да преминем към следващата част, в която всеки разработчик изпълнява възложената си задача.



От тук нататък упражнението се разделя в три посоки, като всеки разработчик може спокойно да премине директно към частта, която се отнася за него.

5. Разработчик А

Добра практика е всяка задача да се изпълнява на отделен клон, който по-късно съответно се одобрява и събира със основния клон. Затова ще направим нов клон с име “Dev-A” и директно ще преминем на него. За да проверим дали всичко е наред, можем да използваме командата git status.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master)
$ git checkout -b "Dev-A"
Switched to a new branch 'Dev-A'

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-A)
$ git status
On branch Dev-A
nothing to commit, working tree clean
```

Задачите, които Разработчик А получава са следните:

- Да се добави нова опция за нашия калкулатор – деление на числа
- Да се добави нова опция за нашия калкулатор – изваждане с абсолютна стойност (на резултата от изваждането)
- Да се създаде метод, който проверява дали потребителят може да получи достъп към приложението. Достъп се дава при правилно въведена парола, а в противен случай изпълнението на приложението спира. Достатъчно е паролата да се пази като константа в основния ни клас – Startup. За парола задайте “abcd1234”. Преди да се затвори приложението при въведена неправилна парола да се изпише подходящо съобщение.

Примерна имплементация:

В класа Startup:

```
1 reference
private static bool CheckCredentials()
{
    Console.Write("Enter password to gain access: ");
    string password = Console.ReadLine();
    Console.Clear();

    return password == Password;
}


  private const string Password = "abcd1234";
```

В Main метода:

```
bool isAuthorized = CheckCredentials();

if (!isAuthorized)
{
    Console.WriteLine("Access denied.");
    Console.ReadKey( intercept: true);
    return;
}

Console.WriteLine("Console Calculator App");
Console.WriteLine(new string( c: '-',  count: 15));

Console.Write("a = ");
double a = double.Parse(Console.ReadLine() ?? throw new InvalidOpera
```

```
switch (choice)
{
    case "a":
        OptionsManager.Add(a, b);
        break;
    case "s":
        OptionsManager.Subtract(a, b);
        break;
    case "m":
        OptionsManager.Multiply(a, b);
        break;
    case "d":
        OptionsManager.Divide(a, b);
        break;
    case "sabs":
        OptionsManager.SubtractAbs(a, b);
        break;
}
```

В класа OptionsManager:

```
public static string[] OptionsList = {
    "a - Add",
    "s - Subtract",
    "m - Multiply",
    "d - Divide",
    "sabs - Subtract Abs",
};
```

```
1 reference
public static void Divide(double a, double b)
{
    Console.WriteLine($"{a} : {b} = {a / b}");
}

1 reference
public static void SubtractAbs(double a, double b)
{
    Console.WriteLine($"|{a} - {b}| = {Math.Abs(a - b)}");
}
```

След изпълнението на задачите е време промените да бъдат отразени и в GitHub репозиторито. Това ще се случи в новосъздадения клон “Dev-A” и няма да се отрази по никакъв начин на проекта в master клона, който е и основен.

На първо място трябва да добавим промените в локалното ни репозитори с командата git add ., след което ще създадем и изпратим commit. Преди да направим това, можем да проверим текущия статус на локалното ни репозитори с командата git status.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-A)
$ git status
On branch Dev-A
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   GitExercise/OptionsManager.cs
        modified:   GitExercise/Startup.cs

no changes added to commit (use "git add" and/or "git commit -a")
```

Командата връща полезна информация относно състоянието на файловете ни. По файловете OptionsManager.cs и Startup.cs са направени промени, които не са добавени за commit. Друга полезна команда е git diff, която извежда подробна информация за всяка извършена промяна.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-A)
$ git diff
diff --git a/GitExercise/OptionsManager.cs b/GitExercise/OptionsManager.cs
index 6999c0f..28f4f57 100644
--- a/GitExercise/OptionsManager.cs
+++ b/GitExercise/OptionsManager.cs
@@ -7,7 +7,9 @@ namespace GitExercise
         public static string[] OptionsList = {
             "a - Add",
             "s - Subtract",
-            "m - Multiply"
+            "m - Multiply",
+            "d - Divide",
+            "sabs - Subtract Abs",
         };

         public static void Add(double a, double b)
@@ -24,5 +26,15 @@ namespace GitExercise
         {
             Console.WriteLine($"{a} - {b} = {a - b}");
         }
+
+        public static void Divide(double a, double b)
+        {
+            Console.WriteLine($"{a} : {b} = {a / b}");
+        }
+
+        public static void SubtractAbs(double a, double b)
+        {
+            Console.WriteLine($"|{a} - {b}| = {Math.Abs(a - b)}");
+        }
     }
 }
```

След като разгледахме резултата от тези опционални команди е време да се върнем към нашата работа. Ще добавим всички промени и ще създадем локален commit, като зададем подходящо съобщение. След това ще изпратим локалния commit към сървъра.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-A)
$ git add .

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-A)
$ git commit -m "Division, Abs Subtraction added; Introduced authorization"
[Dev-A 06d3bb5] Division, Abs Subtraction added; Introduced authorization
 2 files changed, 39 insertions(+), 1 deletion(-)

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-A)
$ git push
fatal: The current branch Dev-A has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin Dev-A
```

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-A)
$ git push --set-upstream origin Dev-A
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 920 bytes | 920.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for 'Dev-A' on GitHub by visiting:
remote:      https://github.com/it-career/CalculatorApp/pull/new/Dev-A
remote:
To https://github.com/it-career/CalculatorApp.git
 * [new branch]      Dev-A -> Dev-A
Branch 'Dev-A' set up to track remote branch 'Dev-A' from 'origin'.
```

С това за сега приключва работата на Разработчик А.

6. Разработчик Б

Добра практика е всяка задача да се изпълнява на отделен клон, който по-късно съответно се одобрява и събира със основния клон. Затова ще направим нов клон с име “Dev-B” и директно ще преминем на него. За да проверим дали всичко е наред, можем да използваме командата git status.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master)
$ git checkout -b "Dev-B"
Switched to a new branch 'Dev-B'

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-B)
$ git status
On branch Dev-B
nothing to commit, working tree clean
```

Задачите, които Разработчик Б получава, са следните:

- Да се добави нова опция за нашия калкулатор – остатъчно деление (деление, което връща като резултат остатъка)
- Повтаряемост на пресмятанията. Забележете, че след изпълнението на едно пресмятане нашият калкулатор прекъсва изпълнението си. Да се промени така логиката, че потребителят да бъде питан за следващо пресмятане, докато не избере опция за изход.

Примерна имплементация:

В Startup класа – Main():

```
while (true)
{
    Console.Clear();
    Console.WriteLine("Console Calculator App");
    Console.WriteLine(new string( c: '-',  count: 15));

    Console.Write("a = ");

    switch (choice)
    {
        case "a":
            OptionsManager.Add(a, b);
            break;
        case "s":
            OptionsManager.Subtract(a, b);
            break;
        case "m":
            OptionsManager.Multiply(a, b);
            break;
        case "dr":
            OptionsManager.DivideRemainder(a, b);
            break;
        case "ex":
            Console.Clear();
            Console.WriteLine("Goodbye");
            Console.ReadKey( intercept: true);
            return;
    }

    Console.WriteLine("Press any key to continue...");
    Console.ReadKey( intercept: true);
}
```

В класа OptionsManager:

```
public static string[] OptionsList = {
    "a - Add",
    "s - Subtract",
    "m - Multiply",
    "dr - Divide Remainder",
    "ex - Exit"
};
```

```
1 reference
public static void DivideRemainder(double a, double b)
{
    Console.WriteLine($"{a} % {b} = {a % b}");
}
```

След изпълнението на задачите е време промените да бъдат отразени и в GitHub репозиторито. Това ще се случи в новосъздадения клон “Dev-B” и няма да се отрази по никакъв начин на проекта в master клона, който е и основен.

На първо място трябва да добавим промените в локалното ни репозитори с командата git add ., след което ще създадем и изпратим commit. Преди да направим това, можем да проверим текущия статус на локалното ни репозитори с командата git status.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-B)
$ git status
On branch Dev-B
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   GitExercise/OptionsManager.cs
        modified:   GitExercise/Startup.cs

no changes added to commit (use "git add" and/or "git commit -a")
```

Командата връща полезна информация относно състоянието на файловете ни. По файловете OptionsManager.cs и Startup.cs са направени промени, които не са добавени за commit. Друга полезна команда е git diff, която извежда подробна информация за всяка извършена промяна.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-B)
$ git diff
diff --git a/GitExercise/OptionsManager.cs b/GitExercise/OptionsManager.cs
index 6999c0f..d848a4d 100644
--- a/GitExercise/OptionsManager.cs
+++ b/GitExercise/OptionsManager.cs
@@ -7,7 +7,9 @@ namespace GitExercise
         public static string[] OptionsList = {
             "a - Add",
             "s - Subtract",
-            "m - Multiply"
+            "m - Multiply",
+            "dr - Divide Remainder",
+            "ex - Exit"
         };

         public static void Add(double a, double b)
@@ -24,5 +26,10 @@ namespace GitExercise
         {
             Console.WriteLine($"{a} - {b} = {a - b}");
         }
+
+        public static void DivideRemainder(double a, double b)
+        {
+            Console.WriteLine($"{a} % {b} = {a % b}");
+        }
```

След като разгледахме резултата от тези опционални команди е време да се върнем към нашата работа. Ще добавим всички промени и ще създадем локален commit, като зададем подходящо съобщение. След това ще изпратим локалния commit към сървъра.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-B)
$ git add .

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-B)
$ git commit -m "Division with remainder added; Calculations repetition"
[Dev-B 255bd5d] Division with remainder added; Calculations repetition
 2 files changed, 64 insertions(+), 45 deletions(-)
 rewrite GitExercise/Startup.cs (86%)

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-B)
$ git push
fatal: The current branch Dev-B has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin Dev-B
```

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-B)
$ git push --set-upstream origin Dev-B
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 872 bytes | 872.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for 'Dev-B' on GitHub by visiting:
remote:      https://github.com/it-career/CalculatorApp/pull/new/Dev-B
remote:
To https://github.com/it-career/CalculatorApp.git
 * [new branch]      Dev-B -> Dev-B
Branch 'Dev-B' set up to track remote branch 'Dev-B' from 'origin'.
```

С това за сега приключва работата на Разработчик Б.

7. Разработчик В

Добра практика е всяка задача да се изпълнява на отделен клон, който по-късно съответно се одобрява и събира със основния клон. Затова ще направим нов клон с име “Dev-C” и директно ще преминем на него. За да проверим дали всичко е наред, можем да използваме командата git status.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master)
$ git checkout -b "Dev-C"
Switched to a new branch 'Dev-C'

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-C)
$ git status
On branch Dev-C
nothing to commit, working tree clean
```

Задачите, които Разработчик В получава, са следните:

- Да се добави нова опция за нашия калкулатор – степенуване (изписва резултата от повдигането a на степен b)
- Да се добави нова опция за нашия калкулатор – логаритмуване (изписва резултата от логаритъм a с база b)
- Да се добави нова опция за нашия калкулатор – сумата на 2 факториела (изписва резултата от a! + b!)

Примерна имплементация:

В Startup класа – Main():

```
switch (choice)
{
    case "a":
        OptionsManager.Add(a, b);
        break;
    case "s":
        OptionsManager.Subtract(a, b);
        break;
    case "m":
        OptionsManager.Multiply(a, b);
        break;
    case "pow":
        OptionsManager.Power(a, b);
        break;
    case "log":
        OptionsManager.Log(a, b);
        break;
    case "fact":
        OptionsManager.Factorial(a, b);
        break;
}
```

В класът OptionsManager:

```
public static string[] OptionsList = {
    "a - Add",
    "s - Subtract",
    "m - Multiply",
    "pow - Power",
    "log - Logarithm",
    "fact - Sum of 2 factorials"
};
```

```
1 reference
public static void Power(double a, double b)
{
    Console.WriteLine($"{a} ^ {b} = {Math.Pow(a, b)}");
}
```

```
1 reference
public static void Log(double a, double b)
{
    Console.WriteLine($"Log of {a} with base {b} = {Math.Log(a, b)}");
}
```

```
1 reference
public static void Factorial(double a, double b)
{
    long factA = CalculateFact((int)a);
    long factB = CalculateFact((int)b);
    Console.WriteLine($"{a}! + {b}! = {(int)a}! + {(int)b}! = {factA + factB}");
}

2 references
private static long CalculateFact(int a)
{
    long result = a;
    for (int i = a - 1; i >= 1; i--)
    {
        result = result * i;
    }

    return result;
}
```

След изпълнението на задачите е време промените да бъдат отразени и в GitHub репозиторито. Това ще се случи в новосъздадения клон “Dev-C” и няма да се отрази по никакъв начин на проекта в master клона, който е и основен.

На първо място трябва да добавим промените в локалното ни репозитори с командата git add ., след което ще създадем и изпратим commit. Преди да направим това (или след това), можем да проверим текущия статус на локалното ни репозитори с командата git status.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-C)
$ git add .

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-C)
$ git status
On branch Dev-C
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   GitExercise/OptionsManager.cs
        modified:   GitExercise/Startup.cs
```

Командата връща полезна информация относно състоянието на файловете ни. По файловете OptionsManager.cs и Startup.cs са направени промени, които не са добавени за commit. Друга полезна команда е git diff, която извежда подробна информация за всяка извършена промяна.

След като разгледахме резултата от тези опционални команди е време да се върнем към нашата работа. Ще добавим всички промени и ще създадем локален commit, като зададем подходящо съобщение. След това ще изпратим локалния commit към сървъра.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-C)
$ git commit -m "Log, Power, Fact options added"
[Dev-C 38617ab] Log, Power, Fact options added
 2 files changed, 41 insertions(+), 1 deletion(-)
```

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-C)
$ git push
fatal: The current branch Dev-C has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin Dev-C


danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (Dev-C)
$ git push --set-upstream origin Dev-C
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 967 bytes | 483.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'Dev-C' on GitHub by visiting:
remote:      https://github.com/it-career/CalculatorApp/pull/new/Dev-C
remote:
To https://github.com/it-career/CalculatorApp.git
 * [new branch]      Dev-C -> Dev-C
Branch 'Dev-C' set up to track remote branch 'Dev-C' from 'origin'.
```

С това за сега приключва работата на Разработчик В.

8. Сливане на клоновете

Дойде етапът, в който трябва да слеем клоновете на проекта си, за да получим цялата функционалност в главния клон. Нека слеем клоновете подред. Да кажем, че се намираме в следния сценарий: аз съм отговорен да одобрявам или не даден код в проекта. Разработчик А, Б и В са свършили със задачите си и са ми съобщили за това. Сега искам да слея клоновете с техните задачи в главния клон.

Да започнем с клон Dev-A (намирайки се в master):

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master)
$ git merge Dev-A
Updating 5978580..06d3bb5
Fast-forward
 GitExercise/OptionsManager.cs | 14 ++++++++++++-
 GitExercise/Startup.cs        | 26 ++++++++++++++++++++++++++
 2 files changed, 39 insertions(+), 1 deletion(-)

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Резултатът е безпроблемно сливане, защото поначало клон Dev-A тръгва от копие на клон master и само го надгражда. За да завършим сливането можем да commit-нем промените, но ще оставим това за по-късно.

Нека слеем и клон Dev-B с master (в който вече сме слели Dev-A):

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master)
$ git merge Dev-B
Auto-merging GitExercise/Startup.cs
CONFLICT (content): Merge conflict in GitExercise/Startup.cs
Auto-merging GitExercise/OptionsManager.cs
CONFLICT (content): Merge conflict in GitExercise/OptionsManager.cs
Automatic merge failed; fix conflicts and then commit the result.
```

Резултатът е неуспешно автоматично сливане на двата клона – появяват се конфликти. Това е така, защото Dev-B надгражда копие на master, но master клонът се е изменил преждевременно (заради сливането му с Dev-A). Има няколко начина да се разрешат тези конфликти – ние ще го направим през Visual Studio. Като резултат от командата виждаме кои файлове не са успели да се слеят. Нека ги отворим.

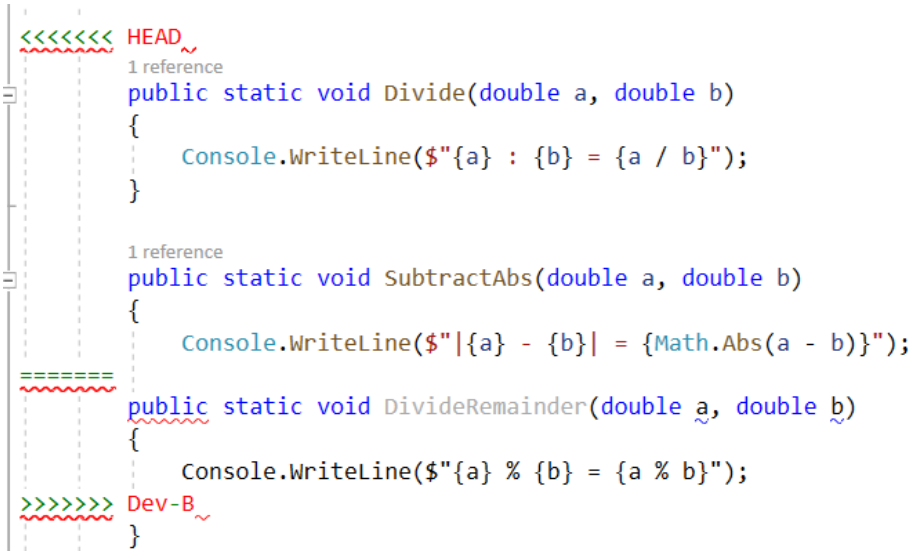
Ако отворим файла OptionsManager.cs, ще открием, че Git е маркирал в коя част от файла са възникнали проблемите. Също е и отбелязано коя версия на кода от кой клон идва:

```
    6 references
    public static class OptionsManager
    {
        public static string[] OptionsList = {
            "a - Add",
            "s - Subtract",
            "m - Multiply",
<<<<<<< HEAD
            "d - Divide",
            "sabs - Subtract Abs",
=======
            "dr - Divide Remainder",
            "ex - Exit"
>>>>>>> Dev-B
        };
```

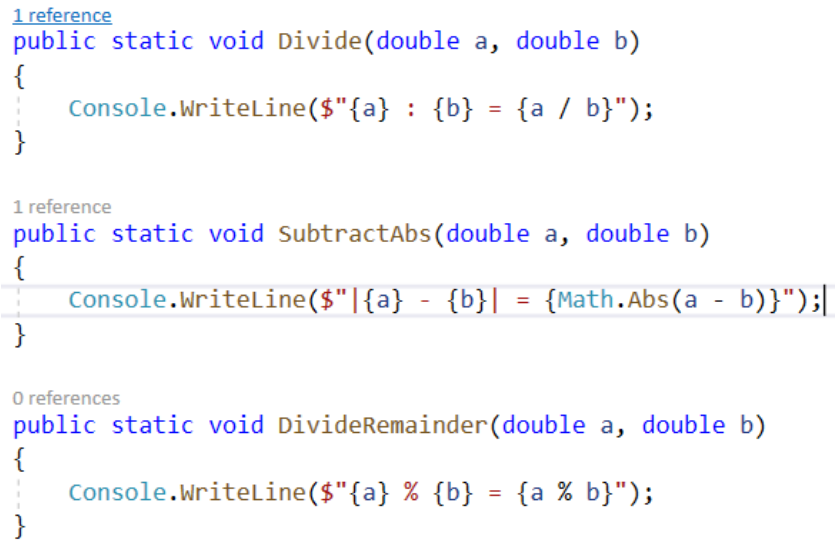
Лесно ще решим този проблем, като оставим и двете добавки към кода:

```
public static string[] OptionsList = {
    "a - Add",
    "s - Subtract",
    "m - Multiply",
    "d - Divide",
    "sabs - Subtract Abs",
    "dr - Divide Remainder",
    "ex - Exit"
};
```

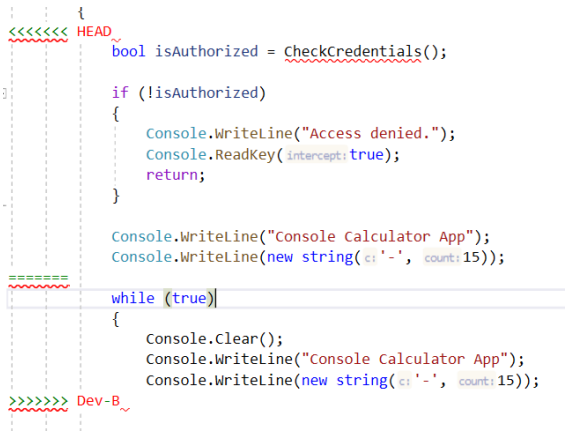
Друг проблем, който е възникнал в този файл е при новосъздадените методи (Git не може сам да реши кои методи да останат – Divide и SubtractAbs или DivideRemainder):



Аналогично ще оставим имплементацията на всички нови методи от двата клона, защото всички са ни нужни:



Да преминем към другия проблемен файл – Startup.cs. Първият конфликт тук е при въвеждането на повтаряемостта на събитията, както и проверката за достъп до приложението:



За да го решим, просто ще оставим както цикъла, така и проверката в началото, а повтарящият се код ще изтрием:

```
0 references
public static void Main()
{
    bool isAuthorized = CheckCredentials();

    if (!isAuthorized)
    {
        Console.WriteLine("Access denied.");
        Console.ReadKey(intercept: true);
        return;
    }

    while (true)
    {
        Console.Clear();
        Console.WriteLine("Console Calculator App");
        Console.WriteLine(new string(c: '-', count: 15));
```

Аналогично ще оставим всички нови методи в switch-case:

```
<<<<<<< HEAD
            switch (choice)
            {
                case "a":
                    OptionsManager.Add(a, b);
                    break;
                case "s":
                    OptionsManager.Subtract(a, b);
                    break;
                case "m":
                    OptionsManager.Multiply(a, b);
                    break;
                case "d":
                    OptionsManager.Divide(a, b);
                    break;
                case "sabs":
                    OptionsManager.SubtractAbs(a, b);
                    break;
            }
=======

            switch (choice)
            {
                case "a":
                    OptionsManager.Add(a, b);
                    break;
                case "s":
                    OptionsManager.Subtract(a, b);
                    break;
                case "m":
                    OptionsManager.Multiply(a, b);
                    break;
                case "dr":
```

, за да получим:

```
switch (choice)
{
    case "a":
        OptionsManager.Add(a, b);
        break;
    case "s":
        OptionsManager.Subtract(a, b);
        break;
    case "m":
        OptionsManager.Multiply(a, b);
        break;
    case "d":
        OptionsManager.Divide(a, b);
        break;
    case "sabs":
        OptionsManager.SubtractAbs(a, b);
        break;
    case "dr":
        OptionsManager.DivideRemainder(a, b);
        break;
    case "ex":
        Console.Clear();
        Console.WriteLine("Goodbye");
        Console.ReadKey( intercept: true);
        return;
}
```

На този етап сме разрешили конфликтите, възникнали от сливането на двата клона. За да ги добавим трябва да ги commit-нем.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master|MERGING)
$ git add .

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master|MERGING)
$ git commit -m "Merge Dev-A-master, Dev-B-master"
[master 9cd3dd9] Merge Dev-A-master, Dev-B-master
```

Време е да слеем и последния клон Dev-C с главния:

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master)
$ git merge Dev-C
Auto-merging GitExercise/Startup.cs
CONFLICT (content): Merge conflict in GitExercise/Startup.cs
Auto-merging GitExercise/OptionsManager.cs
CONFLICT (content): Merge conflict in GitExercise/OptionsManager.cs
Automatic merge failed; fix conflicts and then commit the result.
```

Отново автоматичното сливане е невъзможно. По аналогичен път разрешете възникналите конфликти в кода. След това ще довършим сливането:

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master|MERGING)
$ git add .

danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master|MERGING)
$ git commit -m "Merge Dev-C-master"
[master 9b60715] Merge Dev-C-master
```

Ако на този етап проверим статуса на проекта ще видим дали всичко е наред със сливанията на нашите клонове:

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Ако всичко е наред и получим информация колко commit-а назад е GitHub репозиторито ни в сравнение с локалното, всичко което остава е да изравним двете версии като качим промените.

```
danai@DESKTOP-2GF44TF MINGW64 ~/Desktop/CalculatorApp (master)
$ git push
Enumerating objects: 22, done.
Counting objects: 100% (22/22), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 1.44 KiB | 369.00 KiB/s, done.
Total 10 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), completed with 3 local objects.
To https://github.com/it-career/CalculatorApp.git
   5978580..9b60715  master -> master
```

С това приключва нашата работа по този проект. Към този момент в GitHub репозиторито ни разполагаме с проект, съдържащ цялата функционалност, която имплементирахме. За да тествате дали и в локалното репозитори разполагате с пълната функционалност просто пуснете проекта и вижте дали всеки метод работи, както се очаква.