

שיטות הנדסיות לפיתוח מערכות תוכנה

Acceptance Test

Git

JAVADOC

Acceptance Testing

הגדרה: תהליך בו הלקוח/המשתמש האמיתי של המערכת
בוחן את **הפונקציונליות** של המערכת ורואה שאין שגיאות
או פגמים.

- מה בודקים?
- Input **תקין**
- Input שגוי
- שגיאות, ערכי קצה וארועים
- תנאי קצה
- כל מה שיכול להיות לא תקין!
- כל מה שיכול להיות פגום!

דוגמא: מקרי בדיקה עבור "טופס Login של דף פייסבוק"

- כניסה עם שם משתמש תקין וסיסמה **נכונה**
- כניסה ללא הכנסת שם משתמש וסיסמה
- כניסה רק עם שם משתמש
- כניסה רק עם סיסמה
- כניסה עם שם משתמש שגוי
- כניסה עם סיסמה שגוייה
- כניסה עם שם משתמש שגוי וסיסמה שגוייה

רמת פירוט של הבדיקה:

- רמת הפירוט המקסימלית האפשרית הכתובה בשפה הפשוטה ביותר
- תיעוד וכתיבת שלבי הבדיקה - לא תמיד מי שכתב את הבדיקה גם מבצע אותה
- פירוט הבדיקה יוכל לאשר את ביצוע הדרישה בשלמותה

• טבלה לדוגמא לניהול בדיקות קבלה:

#	Test ID	Description	Expected Result	Precondition	Comments
1	LoginSuccessful	Open "LogIN" window. Type a user ID: 2345236 Type a password: alex124 Click "OK" button.	User logs in. System defines in DB the status of user 2345236 as "logged-in".	User is defined in DB.	The password is correct.
2	LoginWrongPass word	Open "LogIN" window. Type a user ID: 2345236 Type a password: 1234 Click "OK" button.	A system throws out a warning message "User Name/Password incorrect". A system cleans all fields.	User is defined in DB.	The password is not correct.
3	LoginEmptyPass word				
	...				

דוגמא:

בואו נסתכל על הדרישות שזיהינו ונגדיר להן בדיקות קבלה:

Req. number	Requirement description	Req. type (FR or NFR)
1	System allows to add new customer	FR
2	System does not allow to add existing customer	FR
3	System allows to cancel operation	FR
4	"Cancel" operation has to be done by customer	NFR

דוגמא:

בואו נסתכל על הדרישות שזיהינו ונגדיר להן בדיקות קבלה

number	Requirement description	Req. type (FR or NFR)
1	System allows to add new customer	FR
2	System does not allow to add existing customer	FR
3	System allows to cancel operation	FR
4	Operation cancel has to be done by customer	NFR

יש לבדוק שניתן להוסיף לקוח חדש
"Add new customer"

יש לבדוק את המקרה של הוספת לקוח קיים
"Add existing customer"

יש לבדוק את המקרה של ביטול פעולה ע"י הלקוח
"Customer cancels operation"

בדיקות קבלה לדוגמא הנ"ל:

#	Test ID	Description	Expected Result	Precondition	Comments
1	Successful "add new customer"	Open form for entering the customer details. Type a name: Alex Type an e-mail: alex@gmail.com Type an account name: AlexTheKing Type a password: alex124 Click "OK" button.	New customer added to the system. System defines in DB a new row for customer.	Customer doesn't exist in DB.	The new customer was created correctly.
2	Wrong "add new customer"	Open form for entering the customer details. Type a name: Alex Type an e-mail: alex@gmail.com Type an account name: AlexTheKing Type a password: alex124 Click "OK" button.	A system throws out a warning message "User e-mail is incorrect". A system cleans the relevant field.	Customer doesn't exist in DB.	The e-mail is not correct.
3	Add existing customer	Open form for entering the customer details. Type a name: Alex Type an e-mail: alex@gmail.com Type an account name: AlexTheBest Type a password: alex124 Click "OK" button.	A system throws out a warning message "The user is already exists in the DB". A system cleans all fields.	Customer already exists in DB.	
4	Customer cancels operation	Open form for entering the customer details. Type a name: Alex Type an e-mail: alex@gmail.com Type an account name: AlexTheKing Type a password: alex124 Click "Cancel" button.	A system cleans all fields.	Customer doesn't exist in DB.	The customer wasn't added.
	...				

- מה זה git?
- מה זה gitHub?
- מה ההבדלים?

- Git : היא מערכת ניהול גרסאות מבוססת קוד פתוח, שמטרתה לסייע למפתחים בניהול קוד, תיאום עבודה צוותי ומעקב אחר שינויים בקובצי תוכנה.
- מטרותיה העיקריות הן לספק מהירות, שלמות מידע ותמיכה בתהליכים מבוזרים ולא ליניאריים.
- כמערכת בקרת גרסאות מבוזרת, כל ספרייה שלה בכל מחשב נחשבת למאגר נתונים עם תיעוד מלא ואפשרויות מעקב אחר שינויי גרסה, ללא תלות בגישה לרשת או בשרת מרכזי.

- תשימו לב : GitHub ו Git זה לא אותו הדבר.
- Git הוא open-source, version control tool , לעומת GitHub שנותנת לנו כלים אשר ניתן לבצע אינטגרציה עם Git.
- אין לנו צורך ב GitHub על מנת להשתמש ב Git , אבל לא ניתן להשתמש ב GitHub ללא Git.
- אנחנו צריכים את GitHub ל "git-speak as "remotes".

About repositories

- A repository contains all of your project's files and each file's revision history.
- You can discuss and manage your project's work within the repository.
- You can own repositories individually, or you can share ownership of repositories with other people.
- In Git You can restrict who has access to a repository by choosing the repository's visibility.

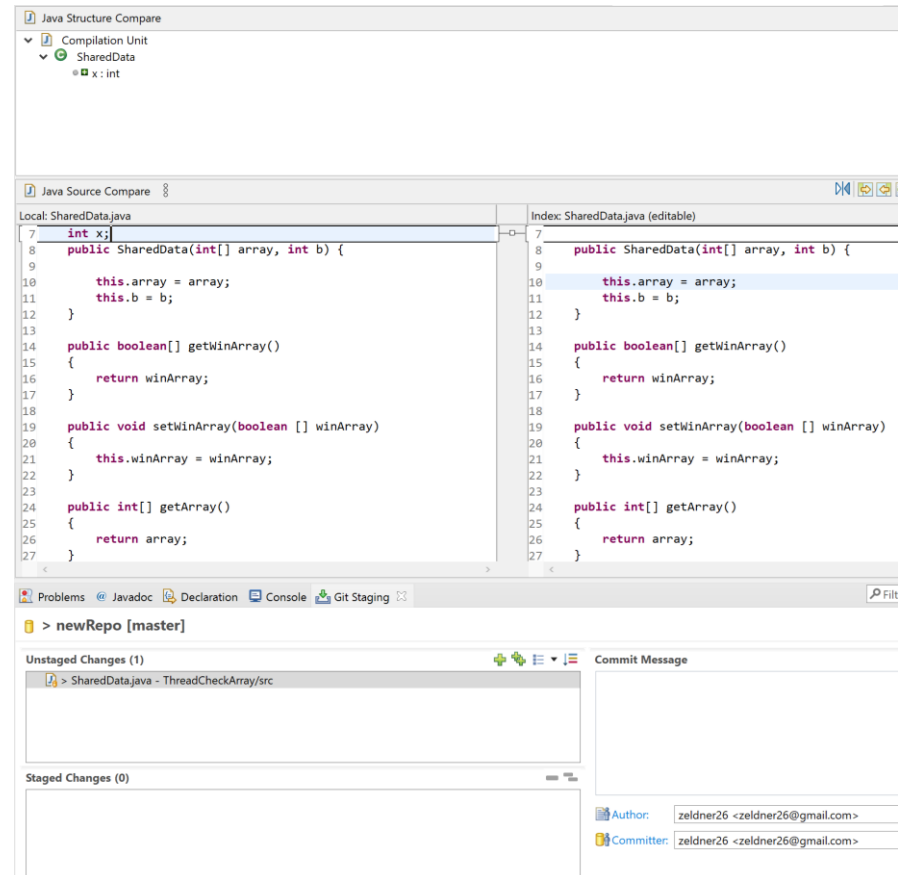
About repositories

- For user-owned repositories, you can give other people collaborator access so that they can collaborate on your project.
- If a repository is owned by an organization, you can give organization members access permissions to collaborate on your repository.
- With GitHub Free for user accounts and organizations, you can work with unlimited collaborators on unlimited public repositories with a full feature set, or unlimited private repositories with a limited feature set.

- You can use repositories to manage your work and collaborate with others.
 - You can use **issues** to collect user feedback, report software bugs, and organize tasks you'd like to accomplish.
 - You can use **discussions** to ask and answer questions, share information, make announcements, and conduct or participate in conversations about a project.
 - You can use **pull** requests to propose changes to a repository.
 - You can use **project boards** to organize and prioritize your issues and pull requests.

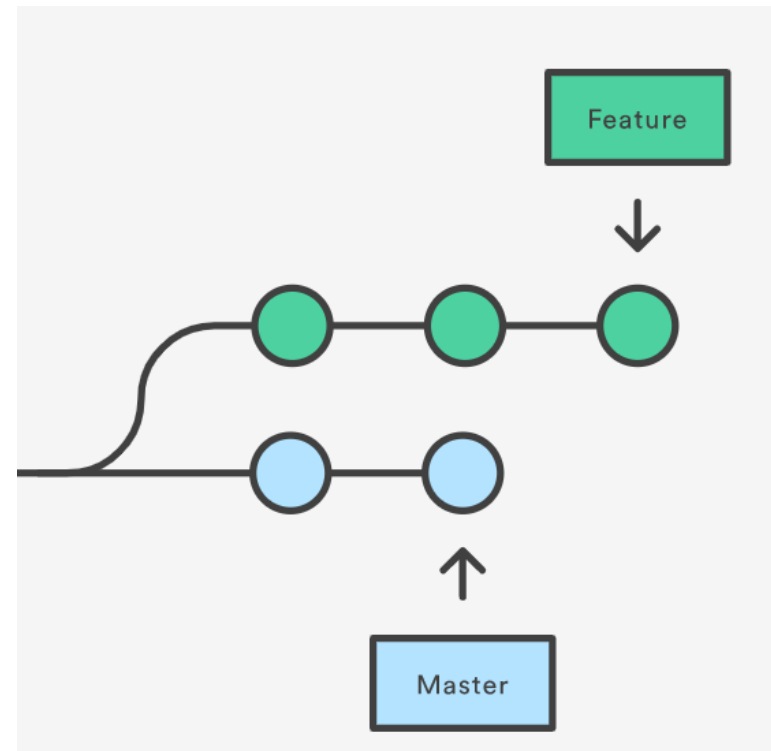
Merging vs. Rebasing

- Both of these commands are designed to integrate changes from one branch into another branch—they just do it in **very different ways**.
- Consider what happens when you start working on a new feature in a dedicated branch, then another team member updates the master branch with new commits.



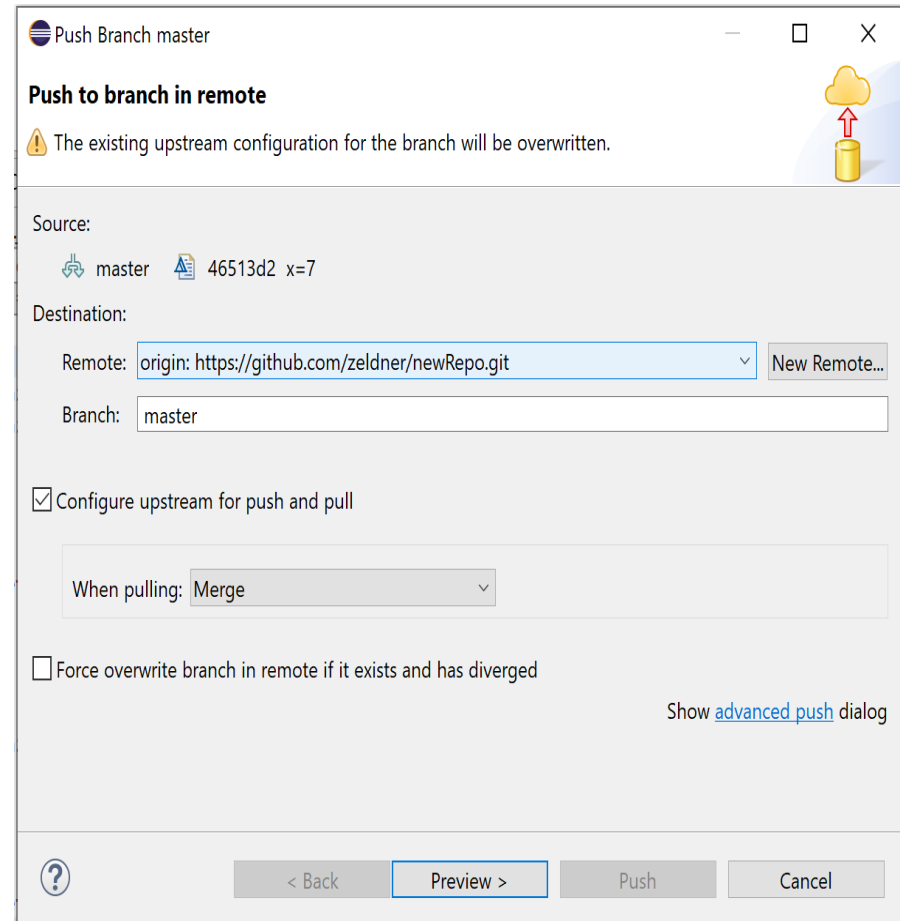
Merging vs. Rebasing

- Now, let's say that the new commits in **master** are relevant to the feature that you're working on.
- To incorporate the new commits into your feature branch, you have two options: **merging or rebasing**.

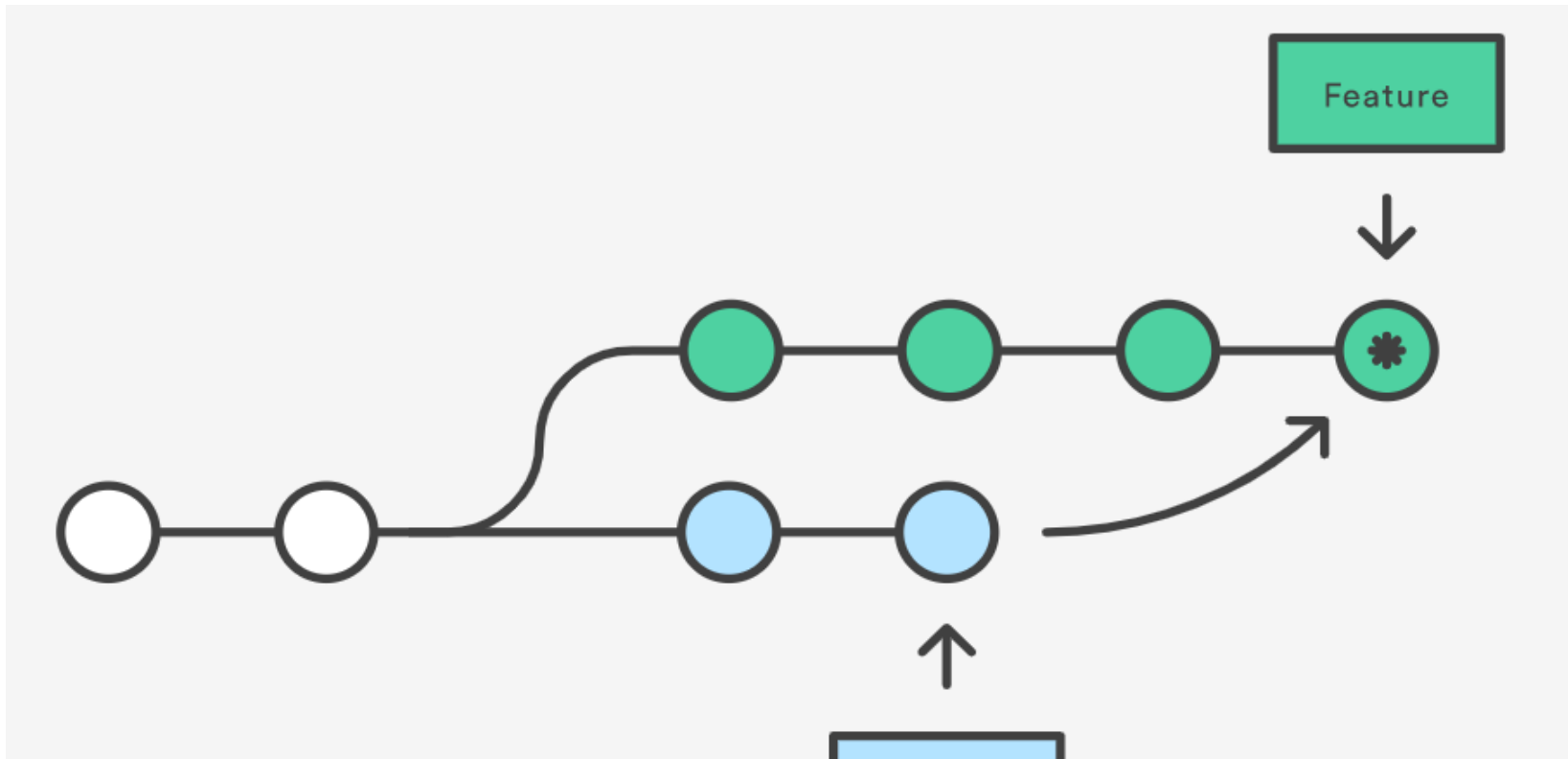


The Merge Option

- The easiest option is to merge the master branch into the feature branch.
- For example :



Merging vs. Rebasing



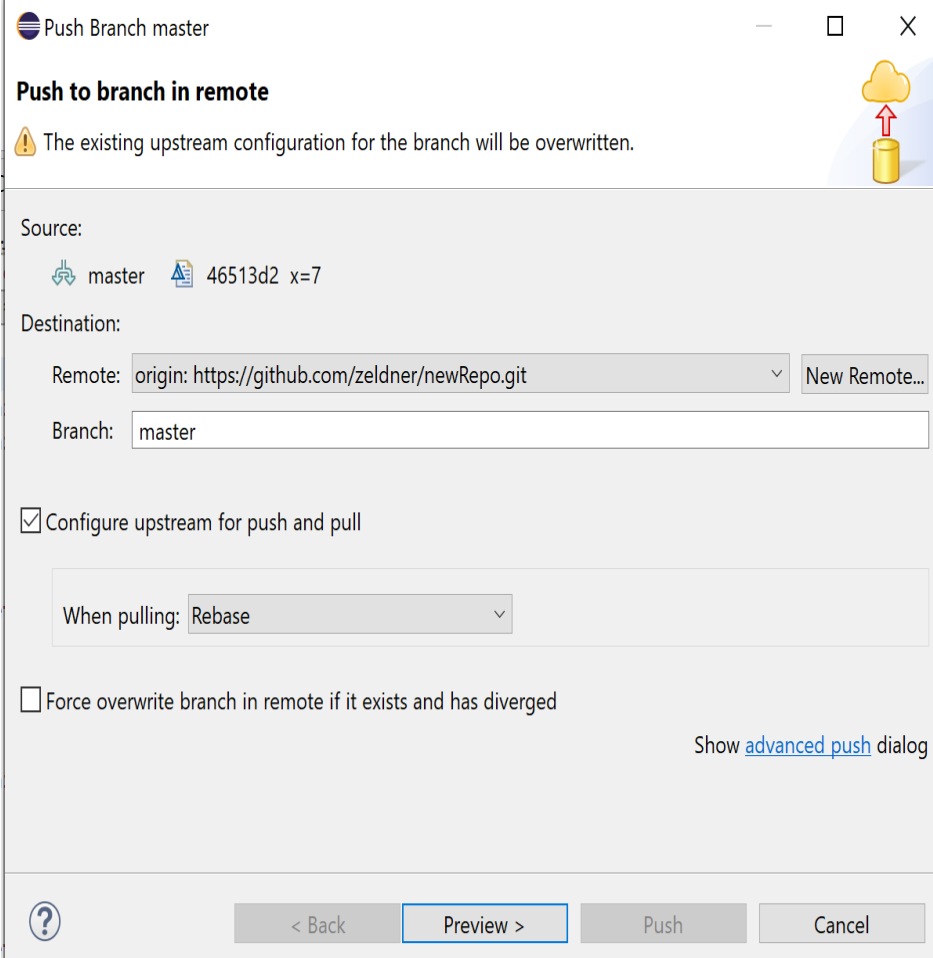
- This creates a new **merge commit** in the feature branch that ties together the histories of both branches, giving you a branch structure that looks like this:

Merging vs. Rebasing

- The existing branches are not changed in any way.
- This avoids all of the potential pitfalls of rebasing.
- On the other hand, this also means that the feature branch will have an extraneous merge commit every time you need to incorporate upstream changes.
- If master is very active, this can pollute your feature branch's history quite a bit.
- While it's possible to mitigate this issue with advanced git log options, it can make it hard for other developers to understand the history of the project.

The Rebase Option

- As an alternative to merging, you can rebase the feature branch onto master branch.
- For example :



Push Branch master

Push to branch in remote

⚠ The existing upstream configuration for the branch will be overwritten.

Source:

🔗 master 46513d2 x=7

Destination:

Remote: origin: https://github.com/zeldner/newRepo.git New Remote...

Branch: master

☒ Configure upstream for push and pull

When pulling: Rebase

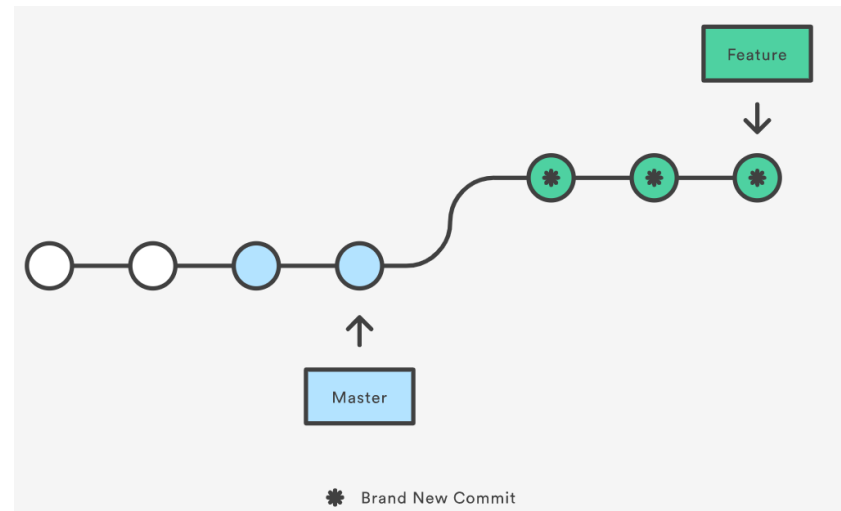
☐ Force overwrite branch in remote if it exists and has diverged

Show [advanced push](#) dialog

? < Back Preview > Push Cancel

The Rebase Option

- This moves the entire feature branch to begin on the tip of the master branch, effectively incorporating all of the new commits in master.
- But, instead of using a merge commit, rebasing *re-writes* the project history by creating brand new commits for each commit in the original branch.

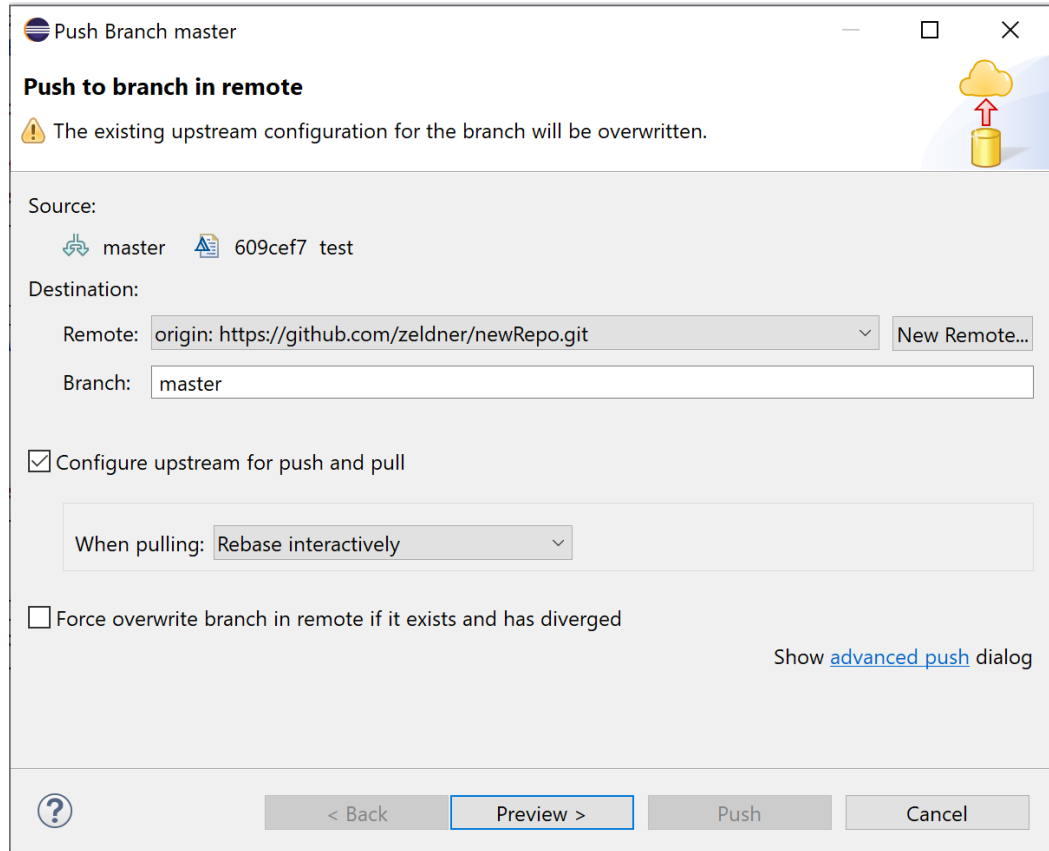


The Rebase Option

- The major benefit of rebasing is that you get a much cleaner project history.
- First, it eliminates the unnecessary merge commits required by git merge.
- Second, rebasing also results in a perfectly linear project history—you can follow the tip of feature all the way to the beginning of the project without any forks.
- This makes it easier to navigate your project with commands.
- But, there are two trade-offs for this pristine commit history: safety and traceability.

Interactive Rebasing

- Interactive rebasing gives you the opportunity to alter commits as they are moved to the new branch.
- This is even more powerful than an automated rebase, since it offers complete control over the branch's commit history.
- Typically, this is used to clean up a messy history before merging a feature branch into master.



Push Branch master

Push to branch in remote

⚠ The existing upstream configuration for the branch will be overwritten.

Source:

📁 master 609cef7 test

Destination:

Remote: origin: https://github.com/zeldner/newRepo.git New Remote...

Branch: master

☒ Configure upstream for push and pull

When pulling: Rebase interactively

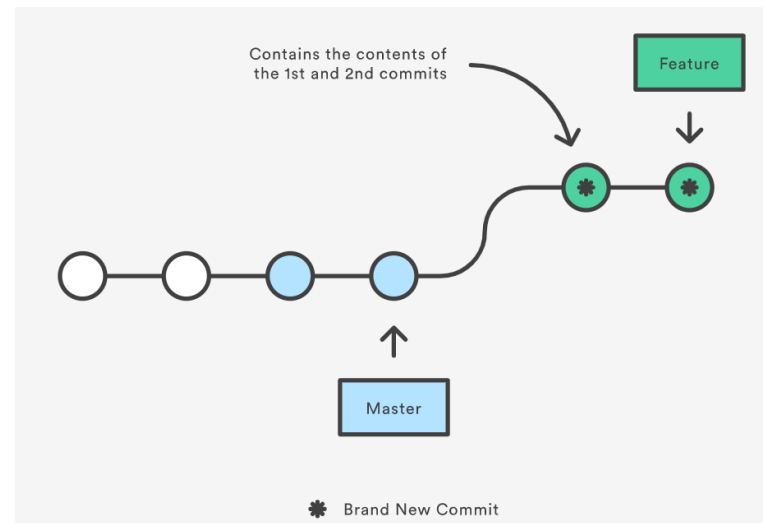
☐ Force overwrite branch in remote if it exists and has diverged

Show [advanced push](#) dialog

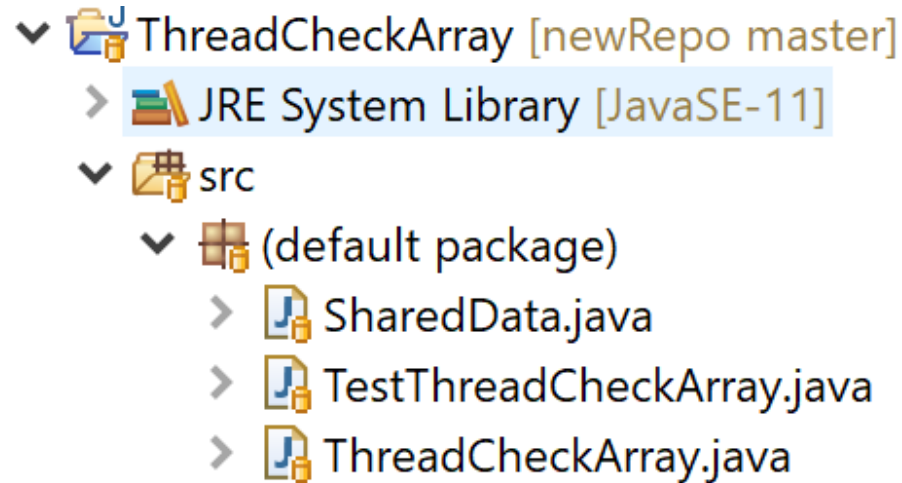
? < Back Preview > Push Cancel

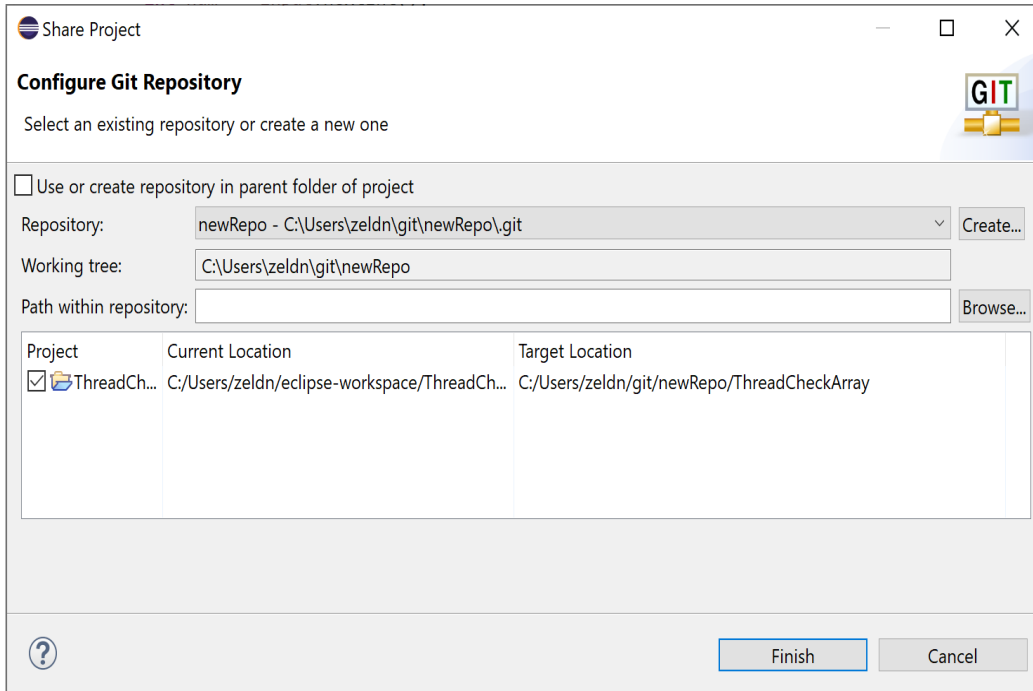
Interactive Rebasing

- Git will perform the rebase according to your instructions, resulting in project history that looks like the following:
- Eliminating insignificant commits like this makes your feature's history much easier to understand.



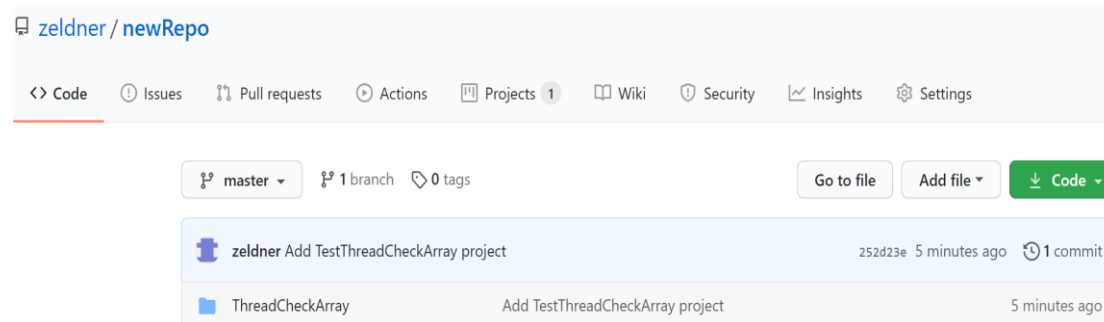
- ראשית יש להרשם באתר של gitHub:
- <https://github.com/>
- מתקינים git ו gitHub באקליפס על פי קובץ ההוראות התקנה.





- מעלים את הפרויקט
TestThreadCheckArray
: כך
- Team > Share Project
- Copy http URL from
GitHub
- For example :

- אחרי שהעלנו ל Git , נבצע על הפרויקט :
- Team > Commit
- Add to index project and Commit and Push to initialize a new Git repository.
- For example :



Managing access to your user account's project boards

- כבעלים של הפרויקט, ניתן להוסיף או להסיר משתף פעולה ולהתאים אישית את ההרשאות שלו לפרויקט ב gitHub
- **הסבר ל צירוף ומחיקת משתף פעולה :**
- <https://docs.github.com/en/github/setting-up-and-managing-your-github-user-account/managing-access-to-your-personal-repositories>
- **הסבר להרשאות לפרויקטים בבעלות המשתמשים :**
- <https://docs.github.com/en/github/setting-up-and-managing-your-github-user-account/permission-levels-for-user-owned-project-boards>

JavaDoc

- ✓ Javadoc is a tool that generates html documentation (similar to the reference pages at java.sun.com) from Javadoc comments in the code.
- ✓ In this tutorial we will go over how to write basic Javadoc comments and how to use features of Javadoc to generate html documentation.

- ✓ Javadoc recognizes special comments `/** */` which are highlighted blue by default in Eclipse (regular comments `//` and `/* ... */` are highlighted green).
- ✓ Javadoc allows you to attach descriptions to classes, constructors, fields, interfaces and methods in the generated html documentation by placing Javadoc comments directly before their declaration statements.

- ✓ Here's an example using Javadoc comments to describe a class, a field and a constructor:

```
/** Class Description of MyClass */
public class MyClass
{
    /** Field Description of myIntField */
    public int myIntField;
    /** Constructor Description of MyClass() */
    public MyClass()
    {
        // Do something ...
    }
}
```


- ✓ **Tags** are keywords recognized by Javadoc which define the type of information that follows.
- ✓ Tags come after the description (separated by a new line).

Here are some common pre-defined tags:

@author [author name] - identifies author(s) of a class or interface.

@version [version] - version info of a class or interface.

@param [argument name] [argument description] - describes an argument of method or constructor.

@return [description of return] - describes data returned by method (unnecessary for constructors and void methods).

@exception [exception thrown] [exception description] - describes exception thrown by method.

@throws [exception thrown] [exception description] - same as ***@exception***.

Here's an example with tags:

```
/** Description of MyClass
 * @author John Doe
 * @author Jane Doe
 * @version 6.0z Build 9000 Jan 3, 1970.
 */
public class MyClass
{
    /** Description of myIntField */
    public int myIntField;
    /** Description of MyClass()
     *
     * @throws MyException          Description of myException
     */
    public MyClass() throws myException
    {
        // Blah Blah Blah...
    }

    /** Description of myMethod(int a, String b)
     *
     * @param a          Description of a
     * @param b          Description of b
     * @return           Description of c
     */
    public Object myMethod(int a, String b)
    {
        Object c;
        // Blah Blah Blah...
        return c;
    }
}
```

Eclipse can generate
Javadoc comments
for classes and
methods.

Place the
cursor in the
text of class or
method
declaration.

Right Click-
>Source->Add
Javadoc
Comment.

Javadoc
comments
with
appropriate
tags are
generated, but
you still have
to write the
descriptions.

Eclipse can also compile Javadocs for projects/packages/classes in the workspace.

Set location of Javadoc command and export your project/package/class as a Javadoc:

- **File->Export.**
- Select **Javadoc** and enter the path of **Javadoc.exe**, i.e. **[Path of J2SE 1.5 SDK]\bin\javadoc.exe** (e.g. **c:\j2sdk1.5.0\bin\javadoc.exe**).
- Also choose your export destination and click **Next**.
- In the **Generate Javadoc Window**, select the project/package(s)/class(es) you want to compile Javadocs for, select the visibility, and enter the path of the destination folder.
- Click **Finish**.

• בהמשך למטלת מטלת Use Case מתרגול 1 :

1. You need to Define acceptance tests for the requirements you have defined of “search” paragraph. (**סימנו בתרגול 1**)

- עליכם לקרוא את הקובץ **Threadq.doc** אשר מסביר את הבעיה ואת הפתרון
TestThreadCheckArray.
- עליכם לשנות את הפתרון, כל שבמקום **int[] array**, הנתונים יהיו ב
ArrayList<Integer> array;
- 2 אנשי הצוות יורידו מה Github את הפרויקט ל Git באקליפס שלהם (כל אחד למחשב שלו)
- על אחד מכם לעדכן את הפרויקט בקובץ **TestThreadCheckArray** ולהעלות את התוצאה ל **GitHub**.
- על השותף השני לעדכן את הקבצים **SharedData** ו **ThreadCheckArray** ולהעלות את התוצאה ל **GitHub**.
- בסיום (לאחר איחוד), על אחד מכם להריץ את הפרויקט ולבדוק שהוא רץ ונותן תוצאה נכונה.

- בגרסאות מסוימות של אקליפס , לא ניתן לקבל אישור לסיסמאות אשר הגדרנו ב **github**.
- במקרה זה , הפתרון ליצור : **personal-access-token**
- **לינק להסבר :**
<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>

- עליכם לתעד על פי ההסברים של Javadoc command : באקליפס בוחרים : **generate element comment**.
- התייעוד יהיה על כל מחלקה וכל פונקציה בפתרון של **SharedData**
- תעשו יצוא ל javadoc לפרויקט, כמו שהוסבר במצגת.
- **בהצלחה!!!**

הסבר קצר לפתרון שהורדתם :

התוכנית מראה , איך בעזרת שני threads ניתן לבצע את 2 הבדיקות על האיבר a_{n-1} (והוא בלבד) במקביל כאשר:

- הראשון שמוצא פתרון מדווח ל-thread השני שפתרון נמצא.
- כל thread בודק אם ה-thread השני מצא פתרון בכל ביצוע של קריאה רקורסיבית.
- כאשר thread מוצא פתרון הוא מדווח דרך מערך שלם או בוליאני chosen מיהם האיברים שנמצאים בפתרון.
- מהרגע שאחד ה-threads מוצא פתרון ומתחיל לרשום אותו ה-thread השני לא ירשום גם הוא על הפתרון.

הפתרון מסתמך על כך , שיתבצעו שני threads בדיוק.

JDBC: Java DataBase Connectivity

