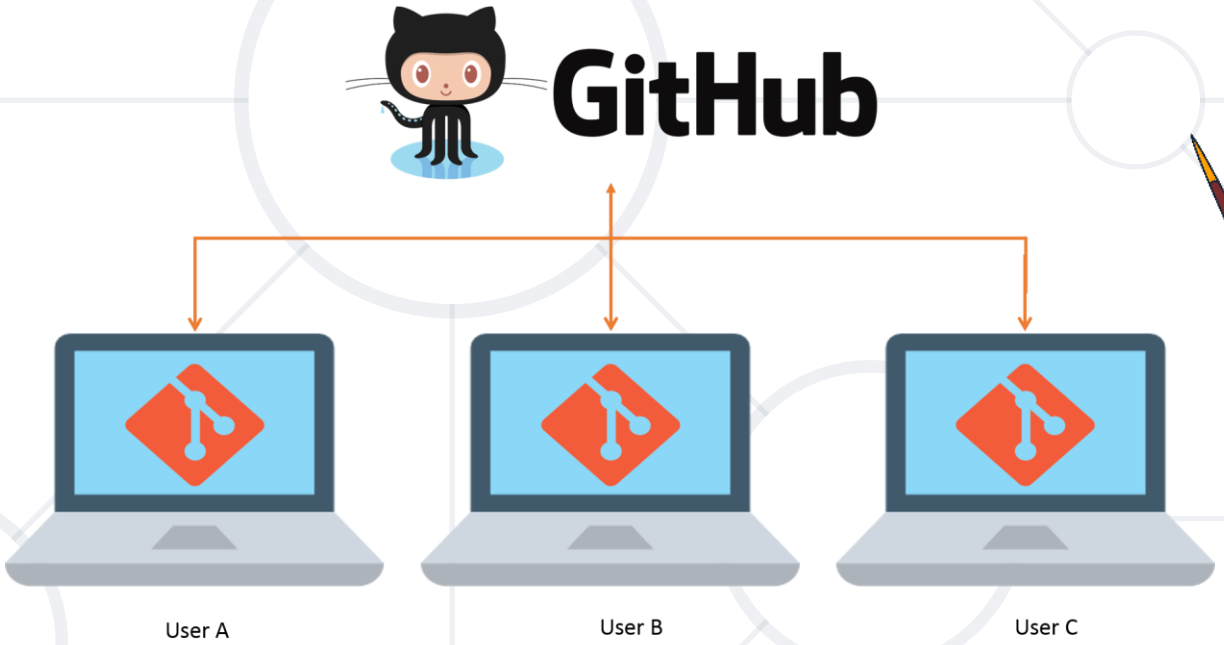
Source Control Systems, Git and GitHub

Working with Git and GitHub



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

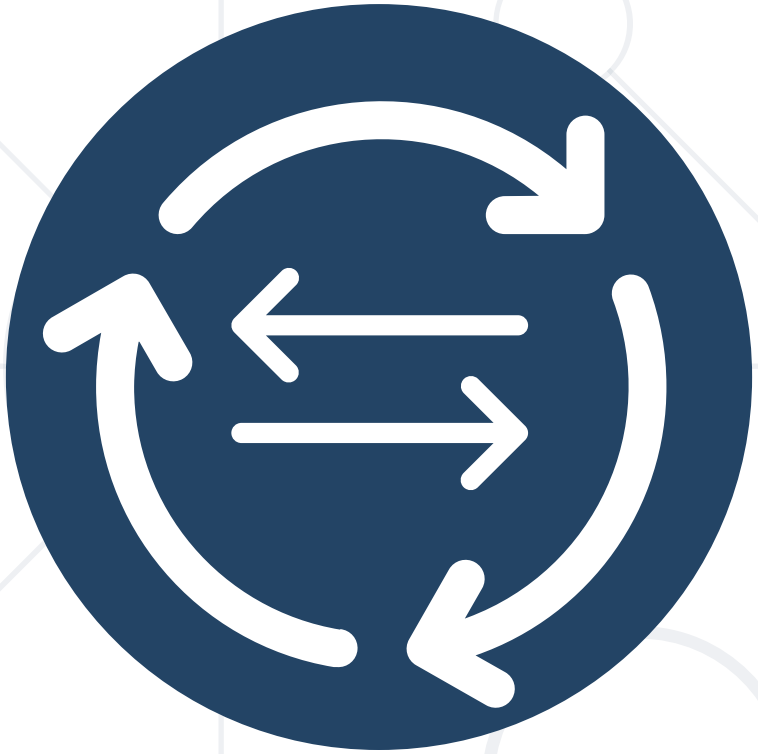
Software University

sli.do

#Dev-Ops

Table of Content

1. Source Control Systems
2. Git
3. GitHub
4. Basic Git Commands
5. Git Conflicts



Source Control Systems

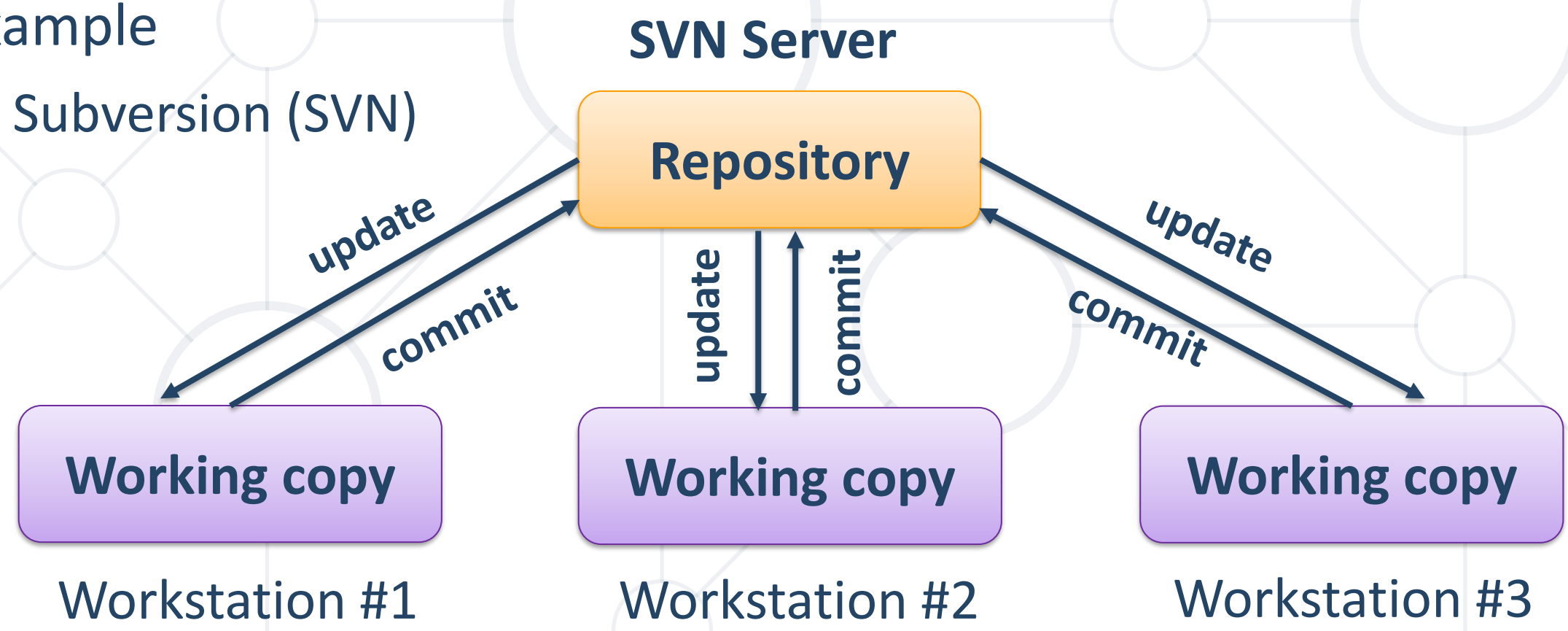
Version Control System

Software Configuration Management (SCM)

- **Version Control System ≈ Source Control System**
  - Tool for **managing** the changes during the **development**
  - A **repository** keeps the **source code** and **other project assets**
    - Keeps a **full history** of **all changes** during the time
  - Solves **conflicts** on **concurrent changes**
- Popular **source control systems**
  - **Git** – distributed source control (hierarchical)
  - Subversion (SVN) – central repository (centralized)

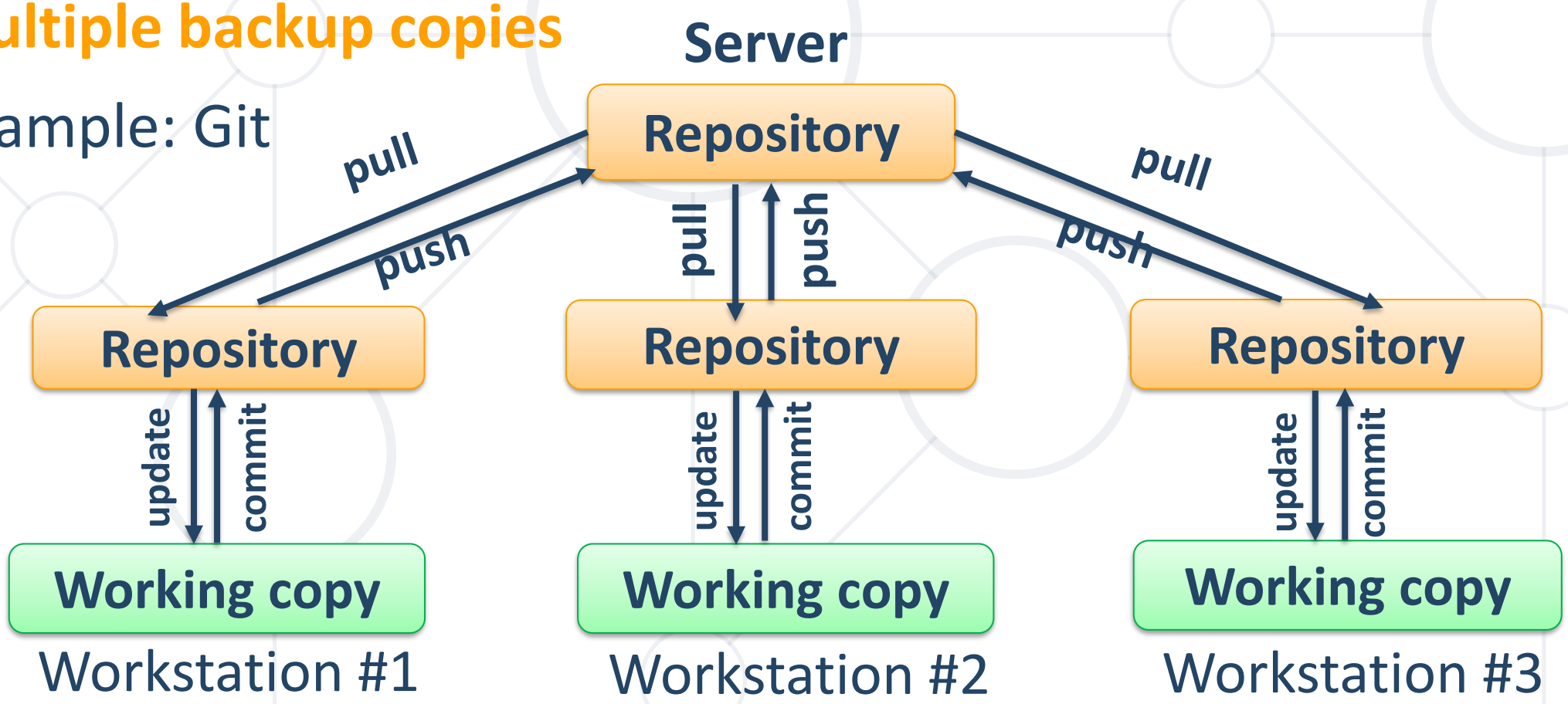
Centralized Version Control (CVC)

- A **CVC system** relies on a **central server** where **developers commit changes**
- Example
  - Subversion (SVN)



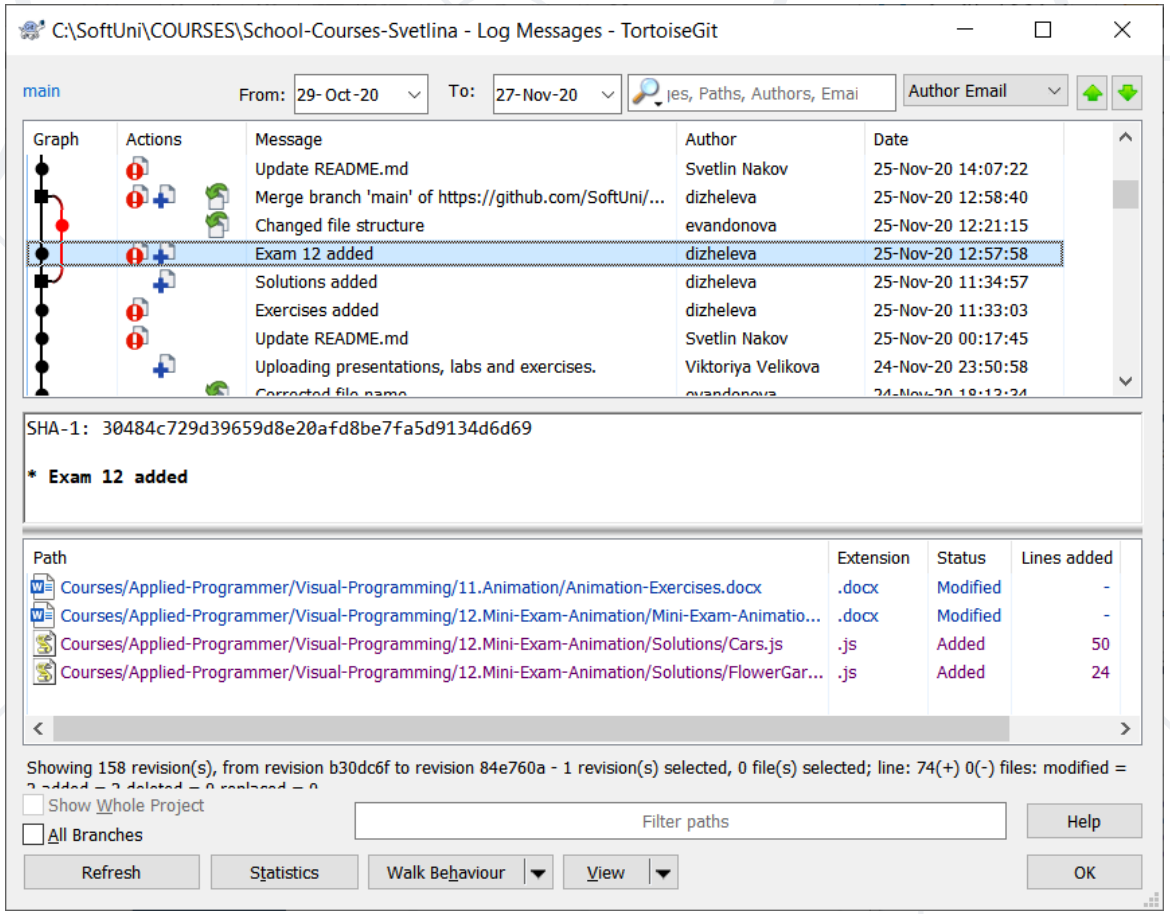
Distributed Version Control (DVC)

- Unlike a CVC system, a DVC doesn't have a **single point** of **failure**
- **Developers clone repositories** on their DVC workstations, creating **multiple backup copies**
- Example: Git



Change Log

- Version control systems keep their own **change log** (version history) and it shows
  - Who?
  - When?
  - What has been changed?
  - Why (purpose)?
- Old versions could be **reverted**



Branching and Branch Management

- **Branching** allows **parallel development** and **isolation** of **changes**
  - Allows working on new features or bug fixes without affecting the main codebase
- Strategies for **branch management**
  - Creating future branches
  - Release branches
  - Hotfix branches
  - Etc.

Git

Global Information Tracker

What is Git?

- Git == distributed **source-control** system
  - The most popular source control in the world
  - **Free**, **open-source** software
  - Works with **local** and **remote repositories**
- Runs on Linux, Mac OS and Windows
  - https://atlassian.com/git/tutorials/install-git
- **GitHub** == **social network** for **developers**
  - **Free project hosting** site with **Git repository**

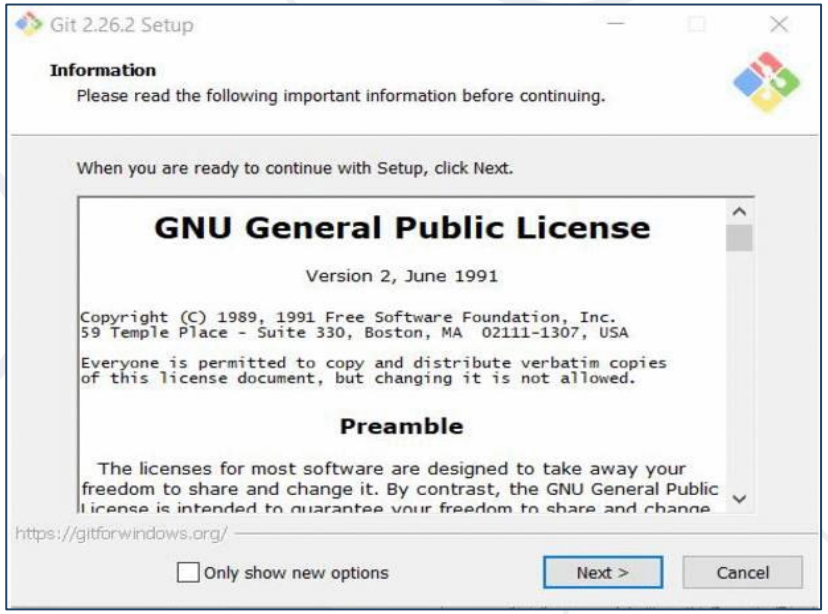
Git Benefits

- Branching
- Clear History
- Collaboration
- Workflow
- Local Repository
- Staging Area
- Distributed
- Trendy
- Integrity
- Speed
- Security
- Scalable
- Data Assurance
- Open Source

Installing Git

- Git installation on **Windows**: Git for Windows (msysGit)
  - https://git-scm.com/downloads
  - Options to select (they should be selected by default)
    - "Use Git Bash Only"
    - "Checkout Windows-style, Commit Unix-style Endings"
- Git installation on **Linux**

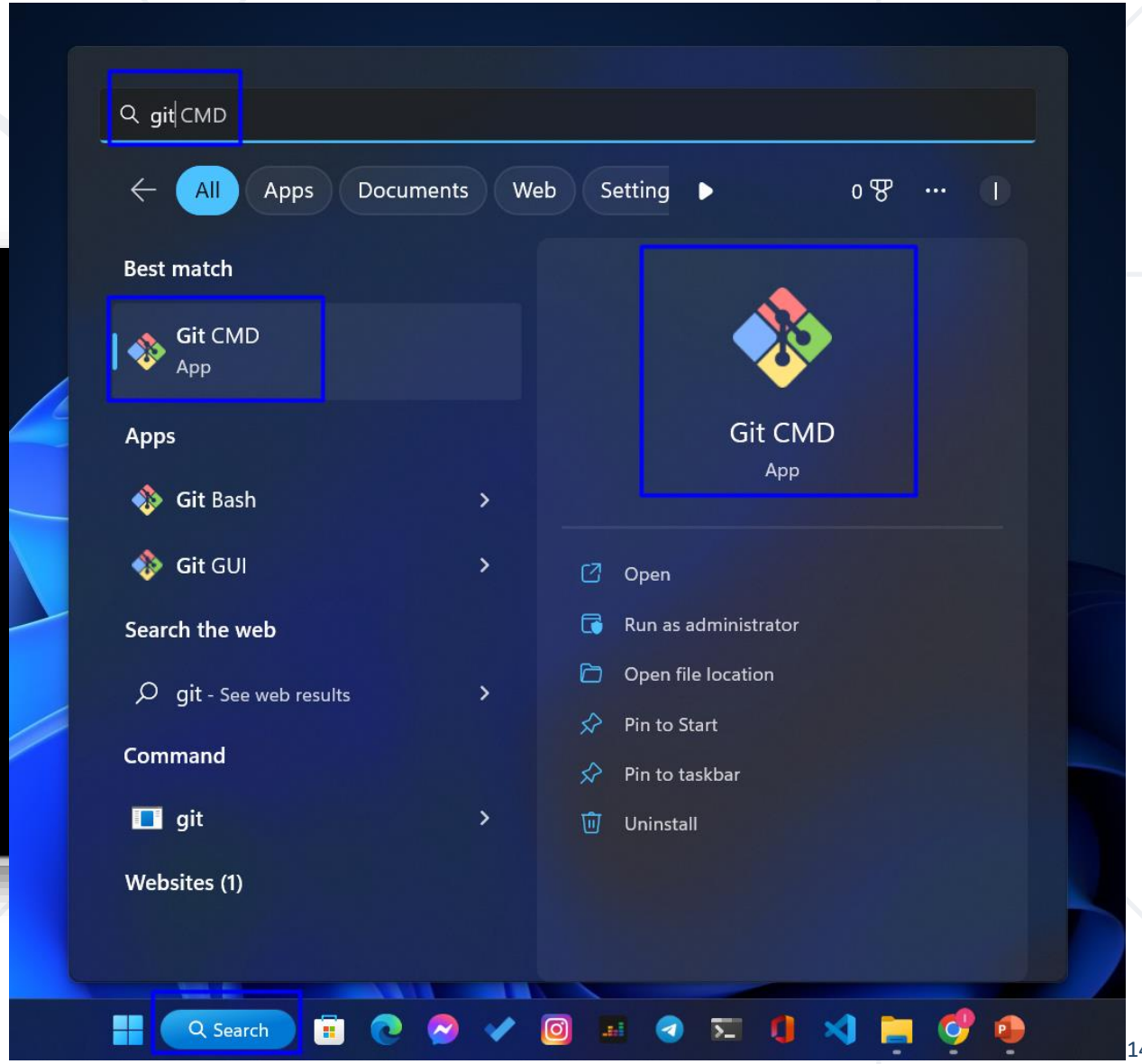
```
sudo apt-get install git
```



Using Git (Command Line)

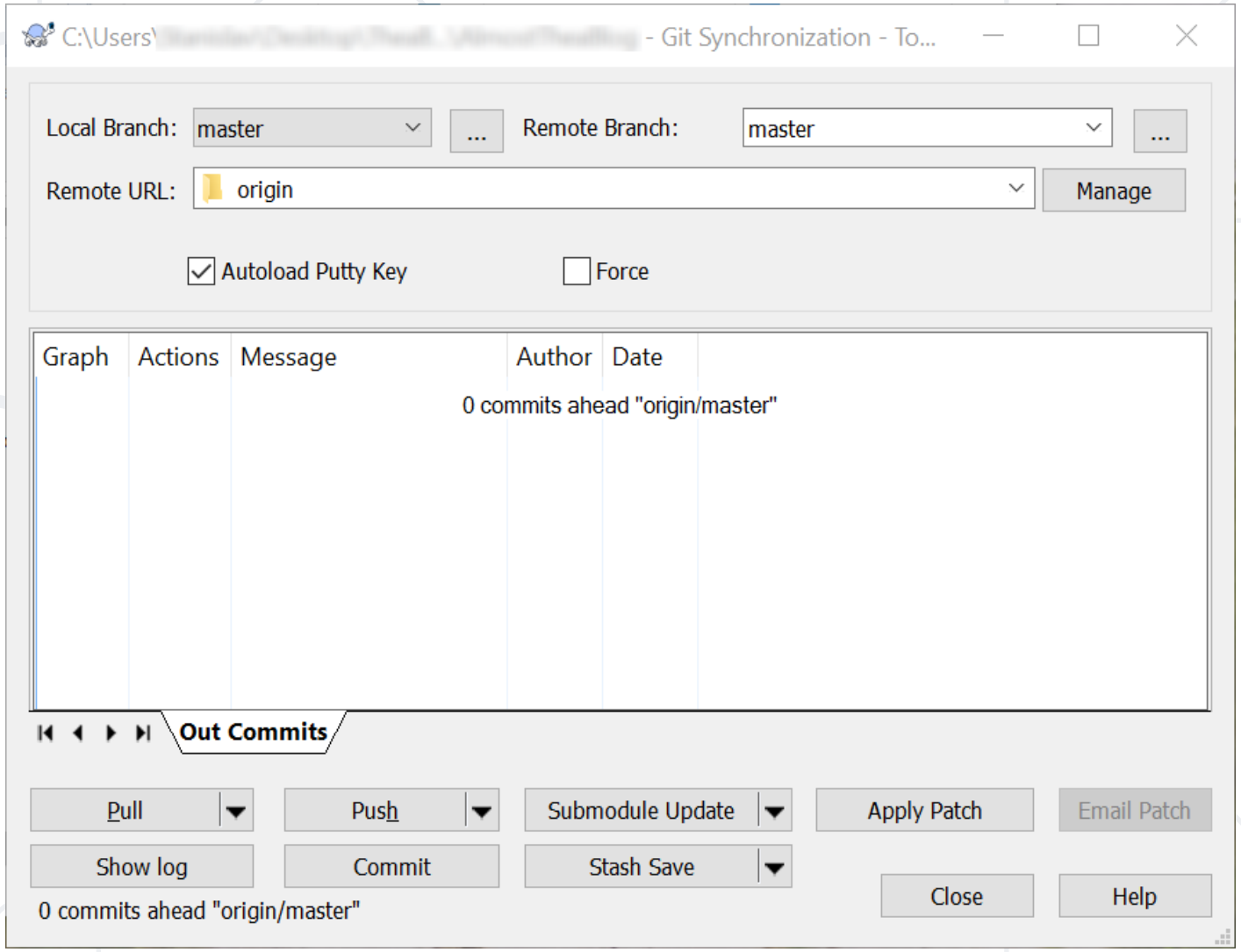
- **GitBash** is a **console-based** client for **git**

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ git checkout master
Switched to branch 'master'

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git merge newbranch
Updating 9e7f7d0..3eb93e9
Fast-forward
 branch file.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 branch file.txt

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$
```



TortoiseGit

- TortoiseGit == Windows Shell Interface to Git (for Windows)
  - Based on TortoiseSVN
  - Simplifies the execution of Git command-line commands using UI



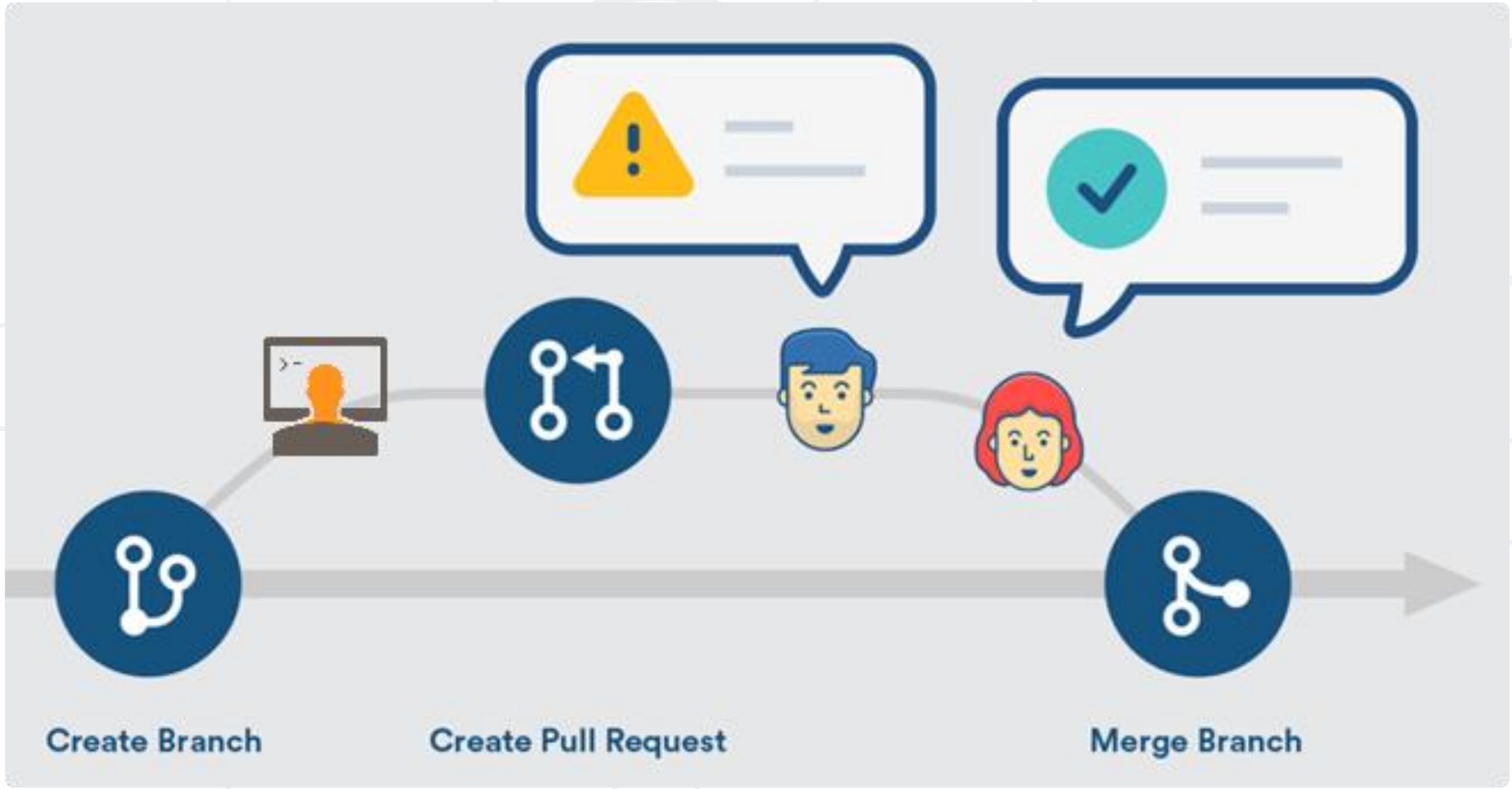
Vocabulary (1)

- **Repo** (repository)
  - Holds the project in a remote server
- **Branch**
  - Parallel development path (separate version of the project)
- **Merge branches**
  - Merge two versions of the same projects

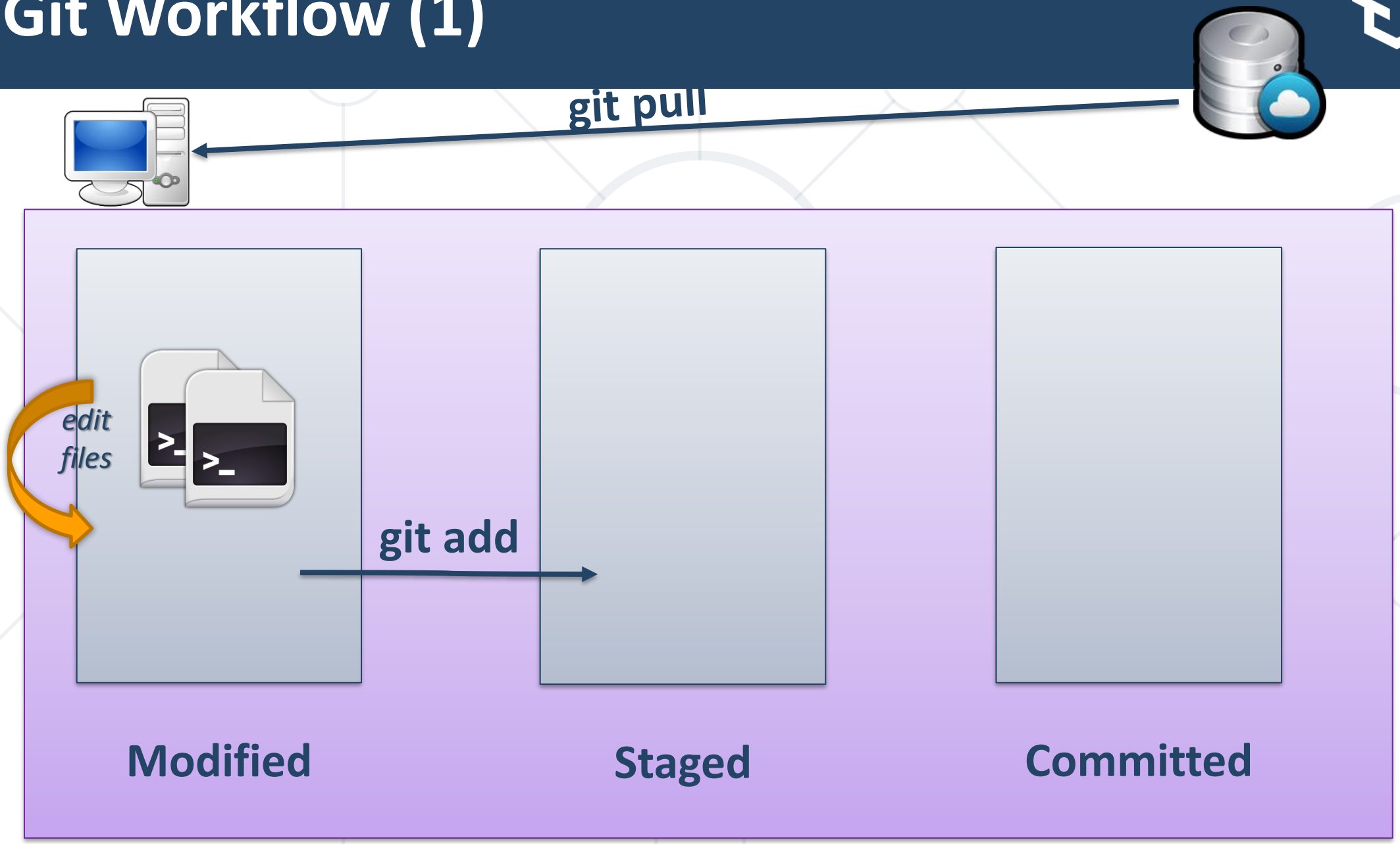
Vocabulary (2)

- **Clone**
  - Download a local copy of the remote project
- **Commit**
  - Saves a set of changes locally
- **Pull**
  - Take and merge the changes from the Remote
- **Push**
  - Send local changes to the Remote

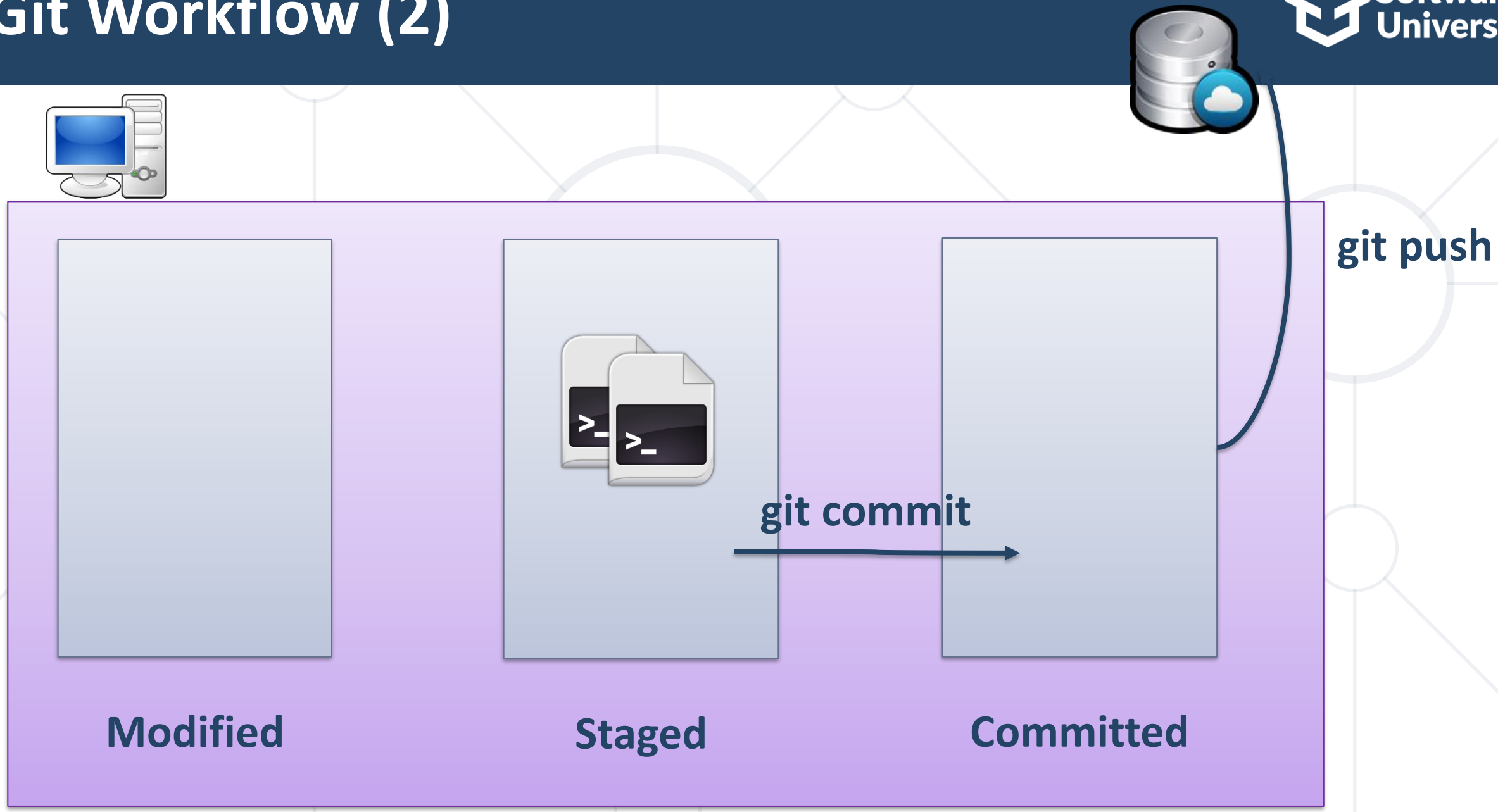
Pull Requests: The Code Review Process



Git Workflow (1)



Git Workflow (2)



GitHub

Source Code Hosting with Git

What is GitHub?

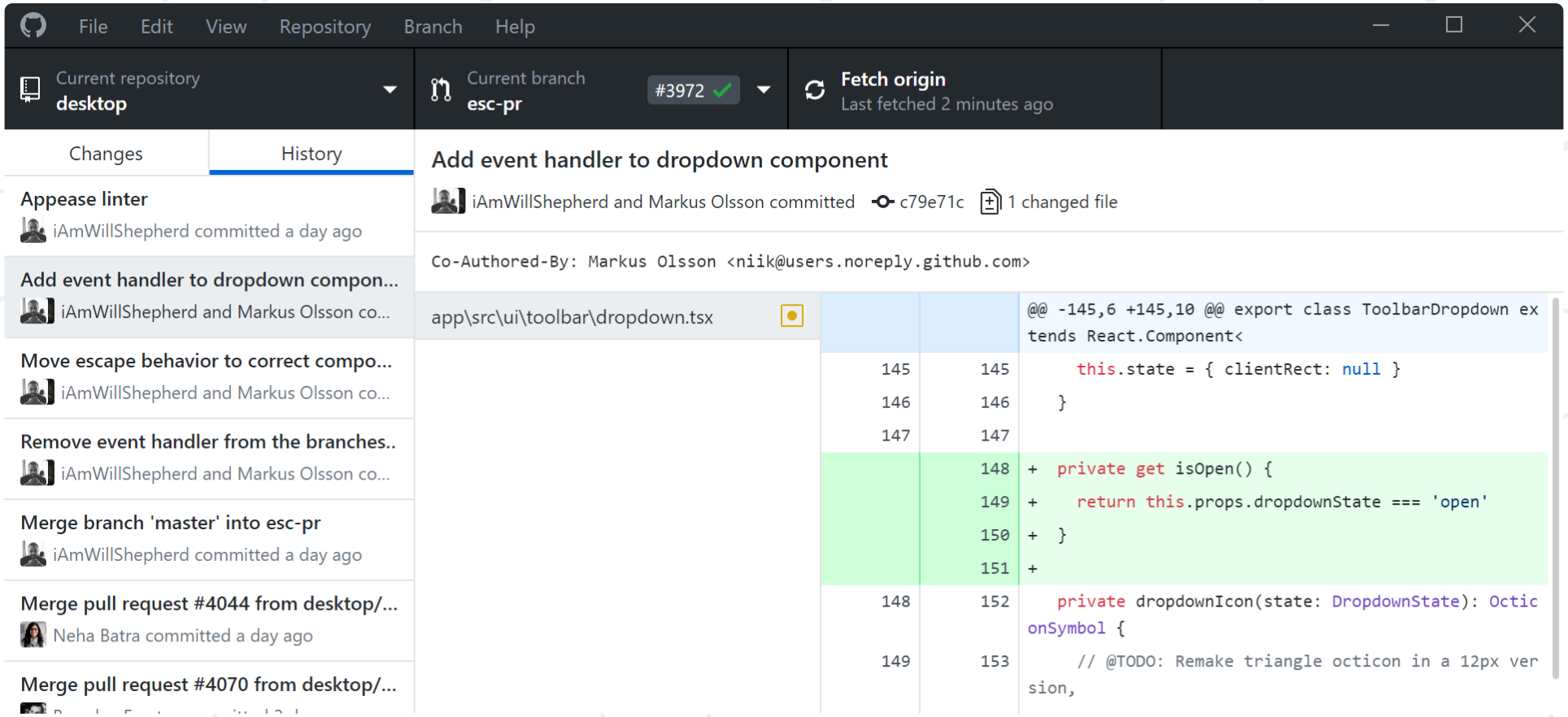
- GitHub == platform and cloud-based service, based on **Git**
- World's **most** used source code host
- Used for **software development** and **version control**
  - Free for **open-source projects** and small private projects
  - Paid plans for private repositories with advanced features

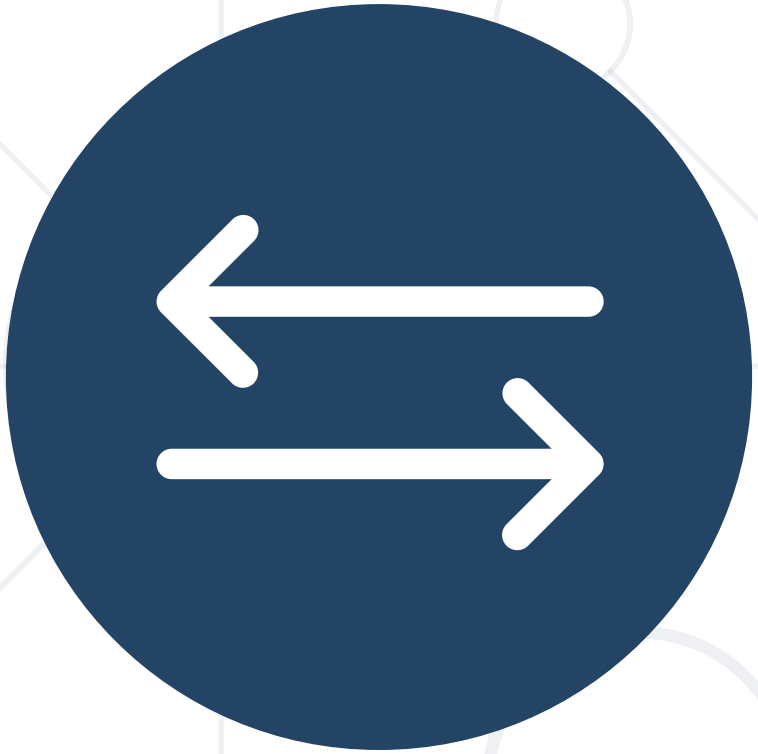
GitHub Features

- **Access control**
- **Bug tracking** (Issue tracker)
- **Continuous Integration** (Actions)
- **Wiki pages** (Documentation)
- **Software feature request**
- **Task management**
- **Project board** (Kanban style)
- Etc.

GitHub Desktop

- **GitHub Desktop** enables interacting with **GitHub** using a **GUI instead** of the **command line** or a **web browser**





Basic Git Commands

Clone ⟶ Modify ⟶ Add ⟶ Commit ⟶ Push

Basic Git Commands (1)

- **Clone** an existing Git repository

```
git clone [remote url]
```

- **Fetch** and **merge** the latest changes from the remote repository

```
git pull
```

- **Prepare** (add / select) files for a commit

```
git add [filename] ("git add ." adds everything)
```

- **Commit** to the local repository

```
git commit -m "[your message here]"
```

Basic Git Commands (2)

- Check the **status** of your local repository (see the local changes)

```
git status
```

- Create a **new** local repository (in the current directory)

```
git init
```

- Create a **remote** (assign a short name for remote Git URL)

```
git remote add [remote name] [remote url]
```

- **Push** to a remote (send changes to the remote repository)

```
git push [remote name] [local name]
```

Example: Using Git Commands (1)

- Open a CLI, for example PowerShell or Terminal
- Clone an existing Git repository

```
git clone https://github.com/SoftUni/playgorund
```

```
PS C:\Users\Desktop\demo> git clone https://github.com/SoftUni/playground
Cloning into 'playground'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 10 (delta 2), reused 3 (delta 1), pack-reused 0
Receiving objects: 100% (10/10), 4.21 KiB | 4.21 MiB/s, done.
Resolving deltas: 100% (2/2), done.
```

- Make **local changes**
- Add (prepare) files for commit

```
git add .
```

Adds everything

```
PS C:\Users\Desktop\demo\playground> git add .
```

Example: Using Git Commands (12)

- Commit added files to the local repository

```
git commit -m "changes"
```

```
PS C:\Users\Desktop\demo\playground> git commit -m "changes"
[main 33630ac] changes
 1 file changed, 1 insertion(+)
 create mode 100644 demo.txt
```

- Push all committed changes to the remote repository

```
git push
```

```
PS C:\Users\Desktop\demo\playground> 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), 312 bytes | 312.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/SoftUni/playground
   9b3bd01..33630ac  main -> main
```

Git Conflicts

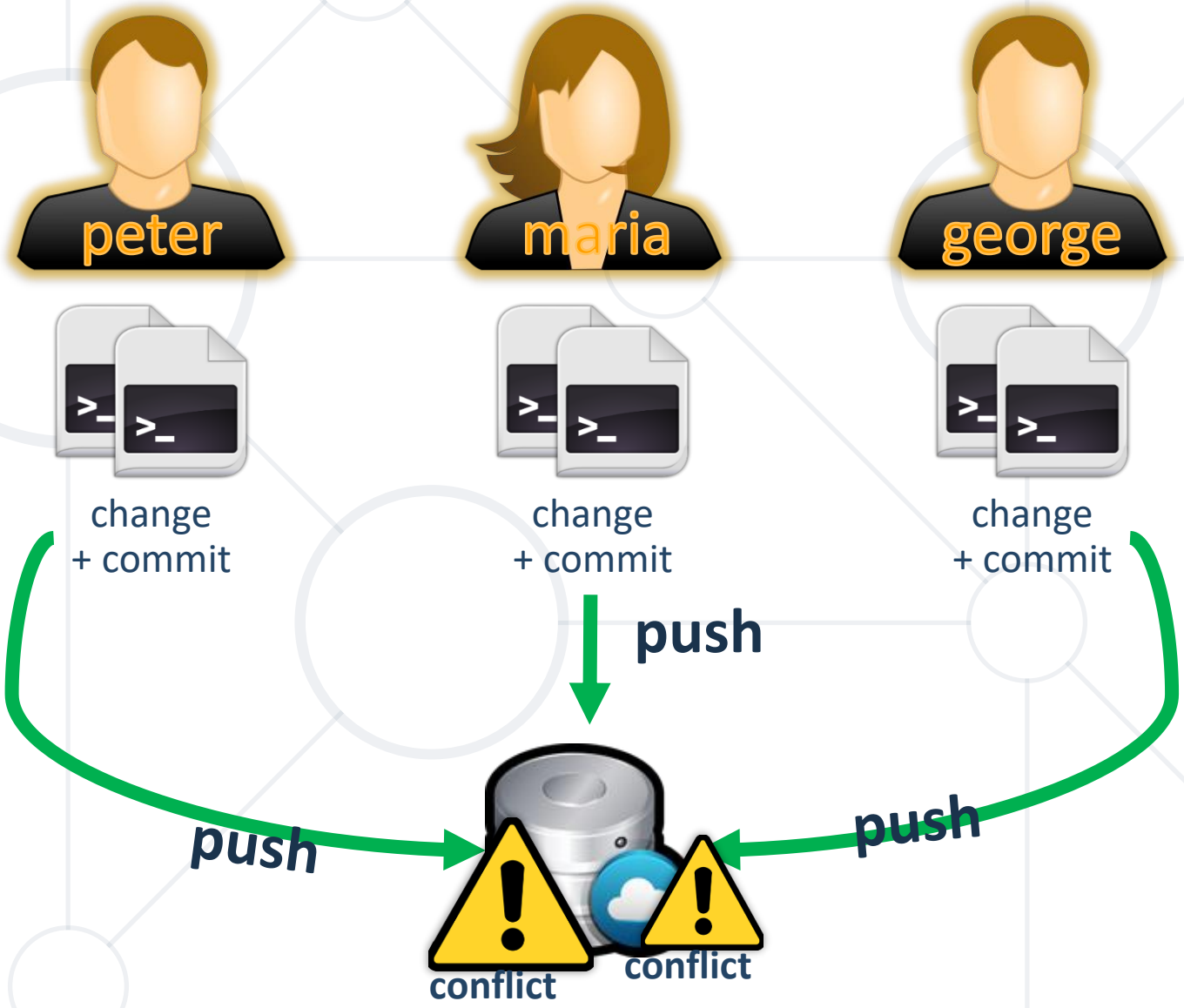
Merging Git Conflicts

Git Conflict

- **Conflicts** generally arise when two or more people **change** the **same file simultaneously**
- Or if a developer **deletes** a **file** while another developer is **modifying** it
- In these cases, **Git cannot automatically determine** what is **correct**
- **Conflicts** only **affect** the **developer conducting** the **merge**
- The rest of the team is **unaware** of the **conflict**

Git Conflict Scenario

- 3 developers work on a **shared project** with Git
  - All of them try to change and push the **same file**
  - A **conflict** will occur on **push**



Competing Line Change Merge Conflicts (1)

- To **resolve** a **merge conflict** caused by **competing line changes**, you must choose which **changes** to **incorporate** in a **new commit**
- Steps to follow
  - Open **Git Bash**
  - Navigate into the **local Git repository** that has the **merge conflict**

```
cd playground
```

```
C:\Users\Desktop\demo>cd playground
```

Competing Line Change Merge Conflicts (2)

- **Display** a **list** of the **files affected** by the **merge conflict**

```
C:\Users\Desktop\demo\playground>git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   demo.txt

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

In this example, the **demo.txt** file has a **merge conflict**

Competing Line Change Merge Conflicts (3)

- **Open** a **text editor** and **navigate** to the **file** with **merge conflicts**
- To see the **beginning** of the **merge conflict** in your file, search the file for the conflict marker **<<<<<<<**
  - You'll see the changes from the **HEAD** after the line **<<<<<<< HEAD**
  - Next, you'll see **=======**, which divides your changes from the changes in the other branch, followed by **>>>>>>> name**

demo.txt - Notepad

File Edit View

```
<<<<<<< HEAD
demo1
=======
demo
test
>>>>>>> 858f78c06b4d3b86924f26bf4579256f206ff3c2
```

Competing Line Change Merge Conflicts (4)

- Decide if you want to keep **only your changes**, keep only the **other changes**, or make a **new change**, which incorporates **both changes**
- **Delete** the **conflict markers <<<<<<<, =======, >>>>>>>** and make the **changes** you want in the **final merge**

demo.txt - Notepad

File Edit View

```
<<<<<<< HEAD
demo1
=======
demo
test
>>>>>>> 858f78c06b4d3b86924f26bf4579256f206ff3c2
```

demo.txt - Notepad

File Edit View

```
demo1
test
```

Competing Line Change Merge Conflicts (5)

- **Add** or stage your changes

```
git add .
```

```
C:\Users\Desktop\demo\playground>git add .
```

- **Commit** your changes with a comment

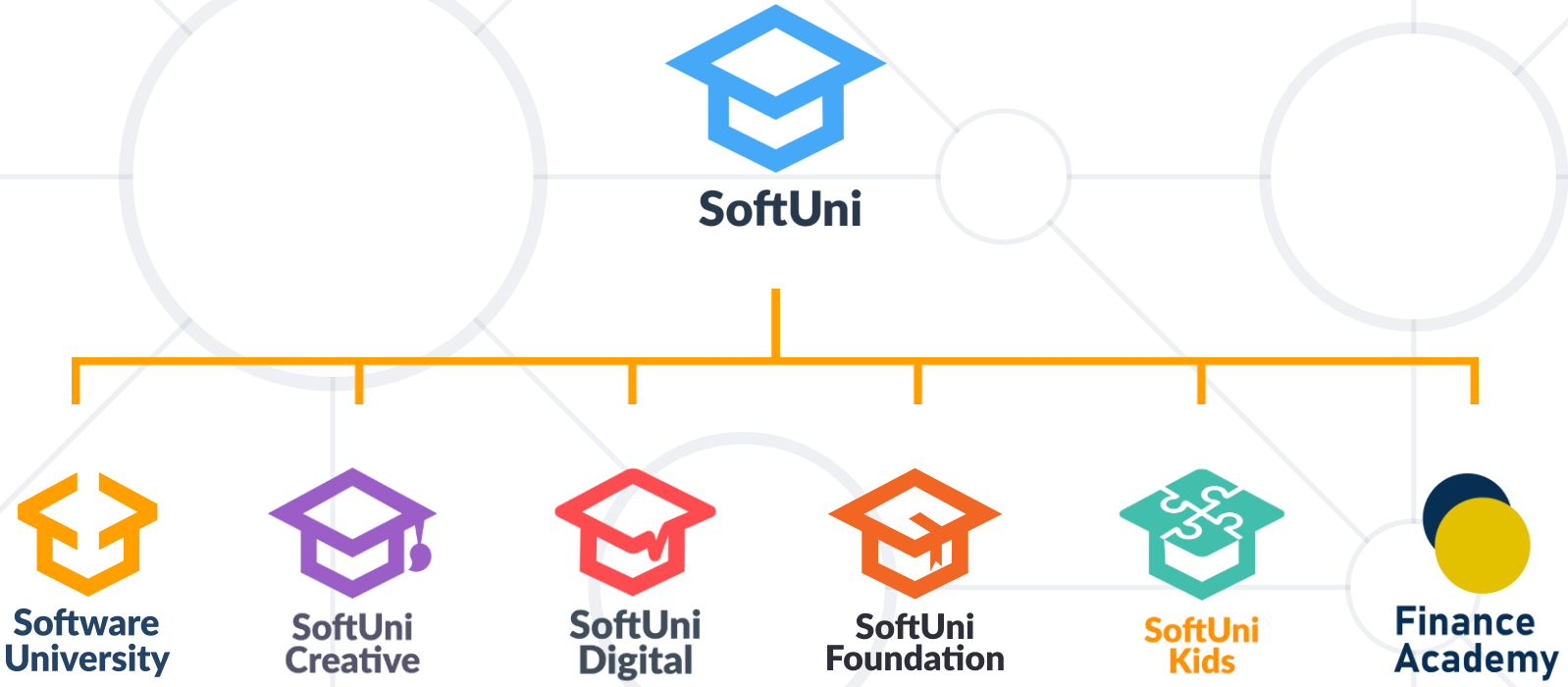
```
git commit -m "Resolved merge conflict."
```

```
C:\Users\Desktop\demo\playground>git commit -m "Resolved merge conflict."
[main 3c8fa03] Resolved merge conflict.
```

Summary

- Use **version control systems** to **work** in a **team**
  - Keep the code in a **central repository**
  - Handle **merge conflicts** with ease
- Git commands: **clone**, **add**, **commit**, **pull**, **push**, **status**, etc.
- **GitHub** == world's #1 code hosting platform
- **Merge conflict** – event that appears when Git is unable to **automatically resolve differences** in **code** between **two commits**

Questions?



© SoftUni – https://about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

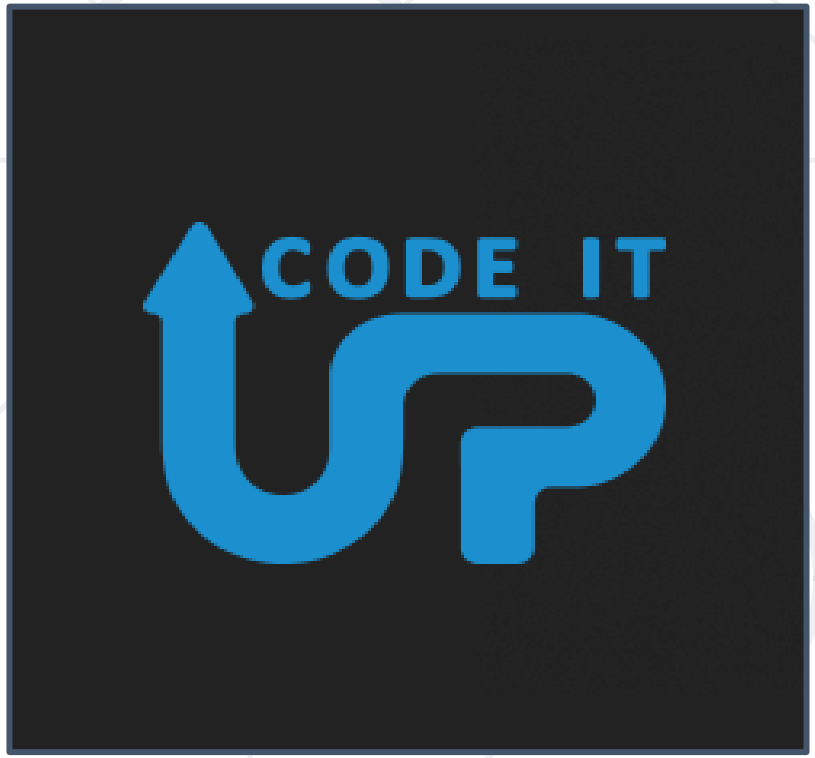
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg