

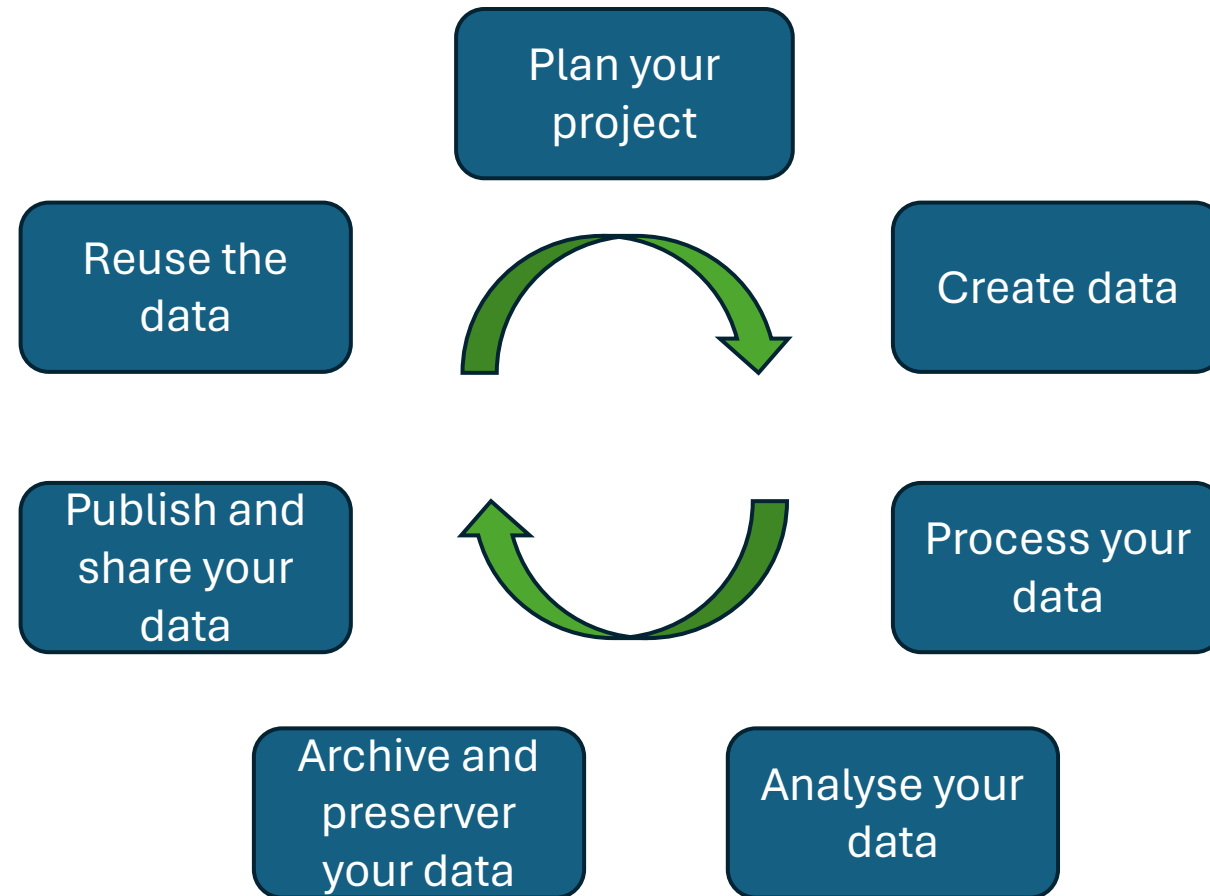


# Data Management

Master Class ‚Digital Scholarly Editing‘ 2024, Saarbrücken  
Saarbrücken, 19.-23.2.2024

Roman Bleier, [roman.bleier@uni-graz.at](mailto:roman.bleier@uni-graz.at)

# Research Data Lifecycle



# FAIR-Principles

# FAIR-Principles

- FAIR Guiding Principles for scientific data management and stewardship (2016)
- FAIR stands for:
  - findability,
  - accessibility,
  - interoperability,
  - reusability
- The goal of these principles is to support good data management
- The principles refer to three types of entities: data (or any digital object), metadata (information about that digital object), and infrastructure

# FAIR: Findability

Metadata and data should be easy to find for both humans and computers

Findability includes:

- Data should be described with rich metadata
- Unique and persistent identifier should identify data and metadata
- It should be an indexed and searchable resource

# FAIR: Accessibility

A user should not only be able to find, but also to easily access data or metadata

Accessibility includes:

- Data should be stored in trusted repositories for long-term preservation
- Should be retrievable by their persistent identifier
- Metadata are accessible, even when the data are no longer available
- Ideally accessible under a well-defined and open licence

# FAIR: Interoperability

The data can be easily integrated with other data. It needs to interoperate with applications or workflows for analysis, storage, and processing.

Interoperability includes:

- Data and Metadata should be available in a formal, accessible, shared, and broadly applicable language
- Metadata and data vocabularies that are used should also follow FAIR principles

# FAIR: Reusability

The ultimate goal of FAIR is to optimise the reuse of data. FAIR data should be ready for reuse in future research.

Reusability involves:

- Data and metadata should be well-described
- Its development well documented
- A clear and accessible license for data reuse
- Metadata and data should meet domain-relevant community standards



# Trusted Repositories

# Why Trusted Repositories?

- The European Union and national funding agencies (DFG, FWF, etc.) request:
- research data follows the FAIR principles
- research data is stored in trusted and certified repositories
- Certificates include CoreTrustSeal or nestor Seal for Trustworthy Digital Archives



# Why Trusted Repositories?

- Certificates help you as researcher to find a good repository for your data
- Certificates ensure that repositories provide certain quality standards
- Repositories following the FAIR principles ensure that your data can be reused in the future

# Licenses

# Why use an open license

- FAIR principles also include concerns about open licenses
- An open license allows your data to be better accessed and reused
- Others can build on your data
- Open licenses provide more visibility to your data
- Because it is reused it will also be more frequently cited

# Creative Commons Licenses

- Creative Commons (CC) licenses is a standardised way to give grant permission for reuse
- A CC-license is a clear message and states what can be done with a resource
- Six (or seven) different license types
- From very open to very restricted

# Creative Commons Licenses

- CC0
  - worldwide public domain, no conditions
- CC BY
  - credit must be given to the creator
- CC BY-SA
  - credit must be given to the creator
  - adoption is possible, but you must license the modified material under identical terms

# Creative Commons Licenses

- CC BY-NC
  - credit must be given to the creator
  - only noncommercial uses of the work are permitted
- CC BY-NC-SA
  - credit must be given to the creator
  - only noncommercial uses of the work are permitted
  - adoption is possible, but you must license the modified material under identical terms



# Creative Commons Licenses

- CC BY-ND
  - credit must be given to the creator
  - no derivatives or adaptations of the work are permitted
- CC BY-NC-ND
  - credit must be given to the creator
  - only noncommercial uses of the work are permitted
  - no derivatives or adaptations of the work are permitted
- <https://creativecommons.org/share-your-work/cclicenses/>

# Git and GitHub

# Some Benefits of Using Git

- Keeps track of files and changes
- Keeps a commit history ( more or less a version history)
- Allows to restore an earlier commit / version
- Great for collaborative work on the same code (software, data, etc.)
- You can fork a repository: meaning that you copy the code from some elses repository and continue to work on your repository
- With services like GitHub and GitLab sharing your code is easy

# Initialising a Git Repository

- **git init**

```
C:\Users\roman\Documents\Projects\IDE-School>git init
Initialized empty Git repository in C:/Users/roman/Documents/Projects/IDE-School/.git/
```

- **git status**

```
C:\Users\roman\Documents\Projects\IDE-School>git status
On branch master

No commits yet

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

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

C:\Users\roman\Documents\Projects\IDE-School>
```

# Adding Files to the Repository

- **git add <FILENAME>:** adds an untracked file to the repository
- **git status:** shows now the files that have been changed

```
C:\Users\roman\Documents\Projects\IDE-School>git add README.md
C:\Users\roman\Documents\Projects\IDE-School>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
       new file:   README.md

C:\Users\roman\Documents\Projects\IDE-School>
```

# Adding Files to the Repository

- **git commit**

- Parameter **-m** can be used for a short commit message
- Git commit -m"commit message between quotes"

```
C:\Users\roman\Documents\Projects\IDE-School>git commit -m"created the readme file"  
[master (root-commit) 472f197] created the readme file  
1 file changed, 3 insertions(+)  
create mode 100644 README.md  
  
C:\Users\roman\Documents\Projects\IDE-School>git status  
On branch master  
nothing to commit, working tree clean
```

# Hands-on

- Install Git on your computer
- Create a folder IDE\_School somewhere
- Open a command prompt and navigate to the folder
- Initialise a Git repository
- Create a README.md file in the folder
- Add the file to your repository and make your first commit

# View changes

## Make changes to the readme file

- **git diff**: the command shows changes between commits and commit and working tree

```
C:\Users\roman\Documents\Projects\IDE-School>git diff
diff --git a/README.md b/README.md
index c37f941..7db2ed9 100644
--- a/README.md
+++ b/README.md
@@ -1,3 +1,7 @@
  ## Readme heading

-This is the text of the readme file
 \ No newline at end of file
+This is the text of the readme file
+
+## New heading
+
+A new line in the file
 \ No newline at end of file
```



# Commit the new changes

## Make a second commit

- **git add .** <DOT> means all changes in the current directory
- **git commit** will create a second commit

```
C:\Users\roman\Documents\Projects\IDE-School>git add .  
  
C:\Users\roman\Documents\Projects\IDE-School>git commit -m"second commit"  
[master 7b6365d] second commit  
1 file changed, 5 insertions(+), 1 deletion(-)  
  
C:\Users\roman\Documents\Projects\IDE-School>|
```

# Working with branches

- git branch: lists all existing branches
- Git branch <BRANCH NAME> makes a new branch

```
C:\Users\roman\Documents\Projects\IDE-School>git branch
* master

C:\Users\roman\Documents\Projects\IDE-School>git branch @roman

C:\Users\roman\Documents\Projects\IDE-School>git branch
@roman
* master
```

# Working with branches

- `git checkout <BRANCH NAME>`: switch to an existing branch

```
C:\Users\roman\Documents\Projects\IDE-School>git branch
@roman
* master

C:\Users\roman\Documents\Projects\IDE-School>git checkout @roman
Switched to branch '@roman'
```

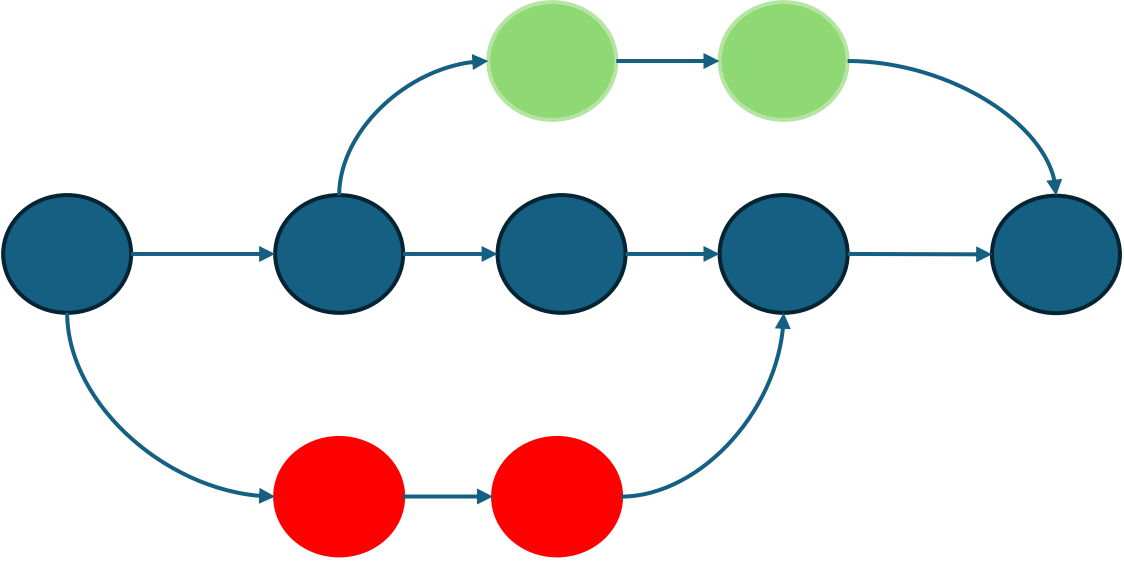
- `git merge <BRANCH NAME>`

```
C:\Users\roman\Documents\Projects\IDE-School>git checkout master
Switched to branch 'master'

C:\Users\roman\Documents\Projects\IDE-School>git merge @roman
Updating 7b6365d..6d0a5c1
Fast-forward
 README.md | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

C:\Users\roman\Documents\Projects\IDE-School>
```

# Working with Branches

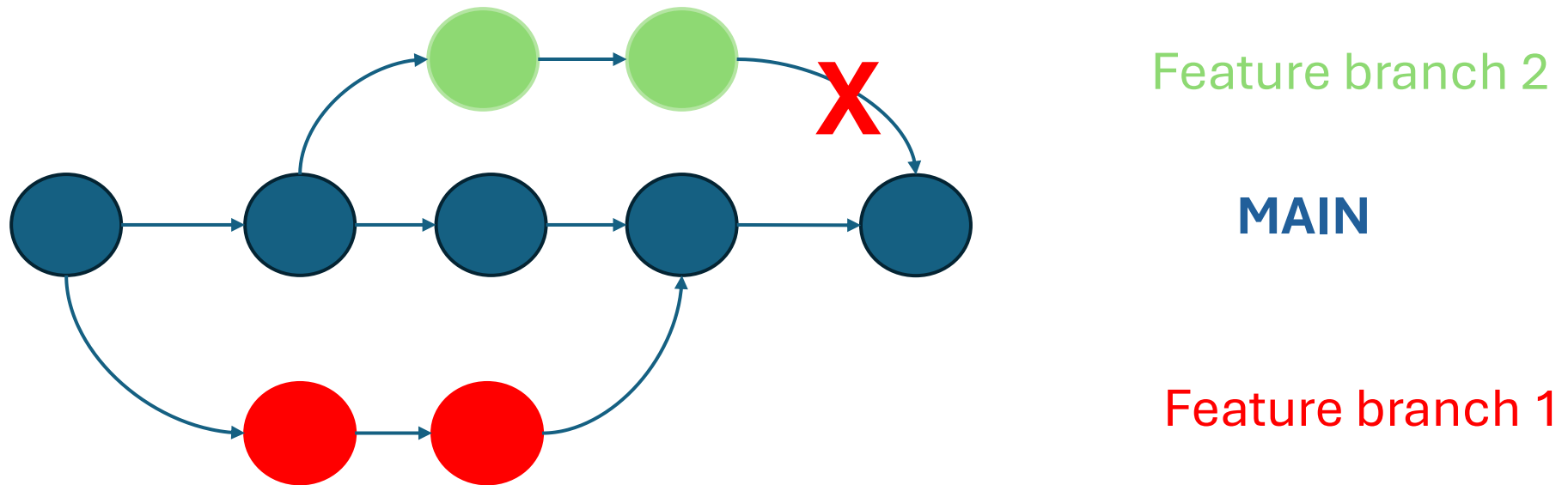


Feature branch 2

**MAIN**

Feature branch 1

# No Worries it is just a Merge Conflict



# No Worries it is just a Merge Conflict

Cause of the merge conflict?

- Usually the same section of code has been changed on two or more branches

What can I do?

- Resolve the merge conflict
- Usually a developer has to look at the section and decide which code should get in the new version

# Hands-on

- Try it yourself
- Create a branch and switch to the branch
- Change the file in the new branch and commit your changes
- Switch back to main / master
- Merge the changes you made in the branch back to main / master

# Sounds all very difficult

- Luckily there are tools that help us with Git-work
  - Help us to keep track of files in the repository and commits
  - Show us our commits and Git-history in a nice and visual way
  - Allow us to easily access earlier commits, etc.
- 
- GitHub provides a client, but there are also other free Git Clients available

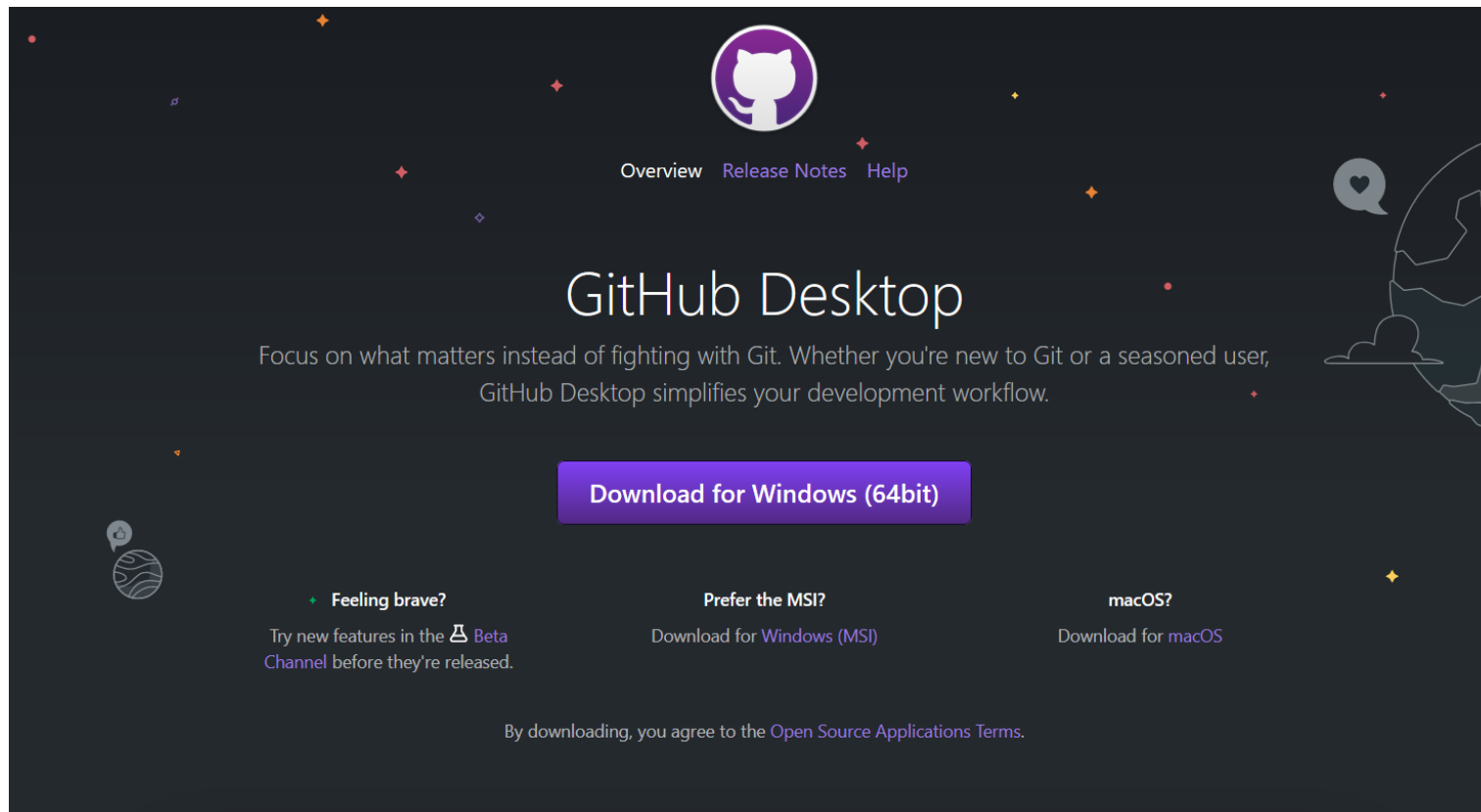


# Github

# Introduction to GitHub

- GitHub is a developer platform that started in 2007
- It allows you to create remote repositories, store, share, etc. your source code
- There is a free plan with certain restrictions, but it is still very good
- Benefits: you can share your data, you can work with colleagues collaboratively
- Github ist not the only such service. Similar services: GitLab, SourceForge, Git Bucket, etc.

# GitHub Desktop Client



# Collaborate and Manage Projects

GitHub provides also other tools for project panning and collaboration

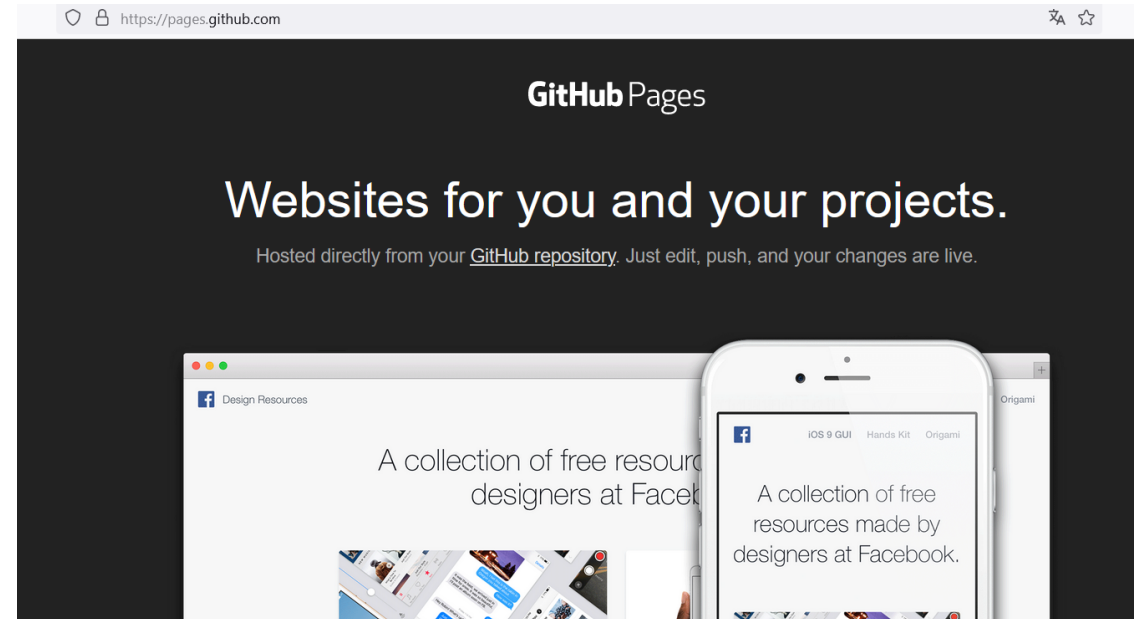
- List of Issues: items that should still be done
- Wiki: for collaborative documentation
- Projects: manage projects on GitHub

# Publish your data and website with GitHub

GitHub is a repository that can be private or public

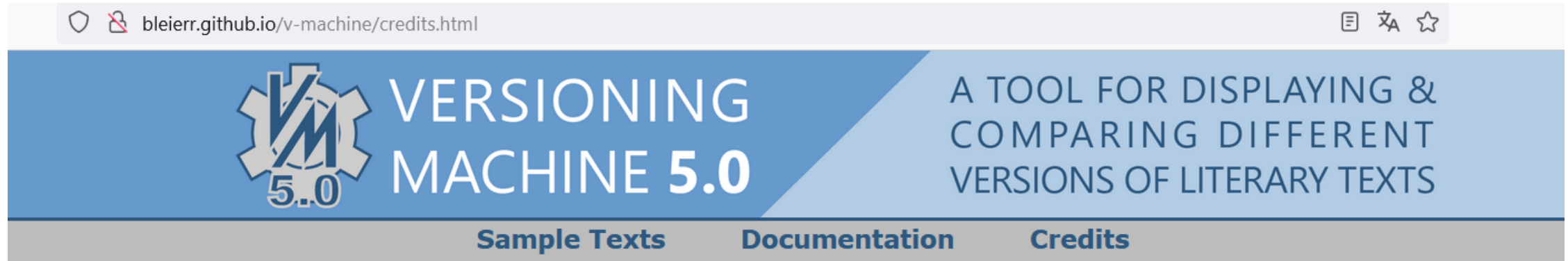
Can be used to share data and program code or work with it in teams

**GitHub pages** provides a means to easily turn your GitHub repository into a website



<https://pages.github.com/>

# Versioning Machine on my GitHub Pages



The screenshot shows the top portion of a web browser displaying the URL `bleierr.github.io/v-machine/credits.html`. The page features a blue header with a logo on the left consisting of a gear with 'VM' and '5.0' inside. To the right of the logo, the text reads 'VERSIONING MACHINE 5.0'. Further right, a description states: 'A TOOL FOR DISPLAYING & COMPARING DIFFERENT VERSIONS OF LITERARY TEXTS'. Below this header is a navigation bar with three links: 'Sample Texts', 'Documentation', and 'Credits', with 'Credits' being the active link.

## Credits

The Versioning Machine was conceived in 2000 by Susan Schreibman, who remains its Founding Editor. She gratefully acknowledges the Digital Humanities Observatory, the University of Maryland Libraries, the Maryland Institute for Technology in the Humanities, and the New Jersey Institute of Technology for their generous support. She also thanks her colleagues, the programmers, designers, and literary scholars, who have so graciously given their time, skills, and energy to the development of the Versioning Machine.

### **The Versioning Machine 5.0 (released December 2015)**

Version 5.0 updated the Versioning Machine to support documents encoded with using the recently released TEI P5 guidelines, as well as greater support for prose documents and images, and various performance enhancements.

### **The Versioning Machine Development Team**

This work is licensed under a [Creative Commons  
Namensnennung 4.0 International License](https://creativecommons.org/licenses/by/4.0/).



All works of other author cited here are their intellectual property and are used for academic purpose only.