Analytical Computing

Lecture 1: Using Git

hogeschool Windesheim

Radboud University

(Her)Introductie

Sebastiaan Ram

22 jaar
Windesheim Software Engineering

Master Data Science
Radboud Universiteit Nijmegen

- Koffie
- Games
- Challenges
- 3D Printing

Agenda

- Analytical Computing
- Git introductie
- Git in de praktijk
- Probeer het zelf
- Git in de CLI

Analytical Computing

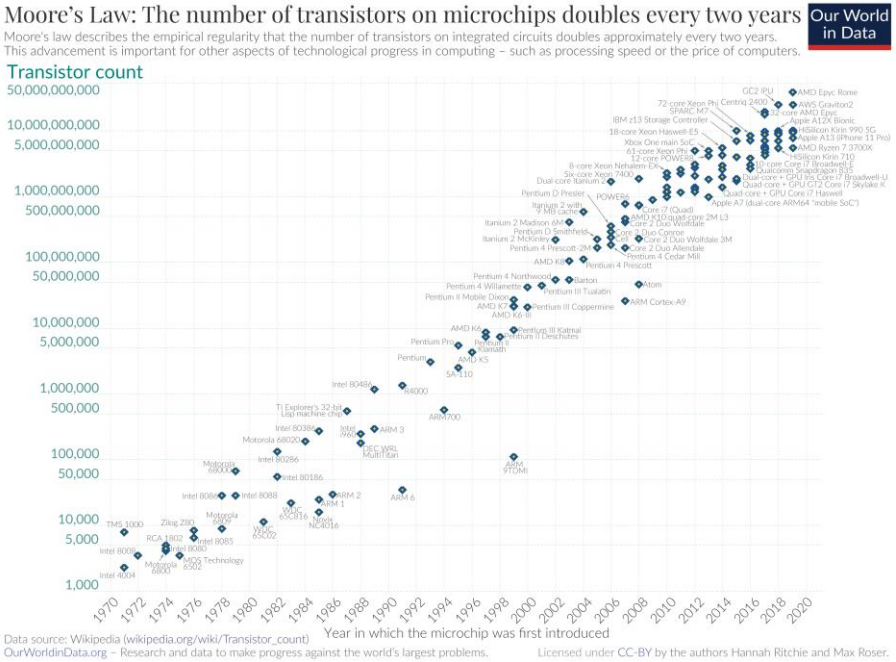
Analytical Computing

- Analysis: “the process of understanding a system quantitatively”
- A diverse collection of people (managers, scientists, etc.) and equally diverse systems (business projects, social entities, etc.) have remarkably similar needs
  - Namely: a repeated cycle of data acquisition and management, data analysis, and presentation of results.

https://link.springer.com/chapter/10.1007/978-3-642-51461-6_2

Analytical Computing

- Computers are the most powerful tool for extracting meaningful insights from data
  - …and they're only getting cheaper
  - …and more powerful
- Moore's Law
  - Number of microchip transistors doubles every **two** years



Planning

20-4-2021 **Git**

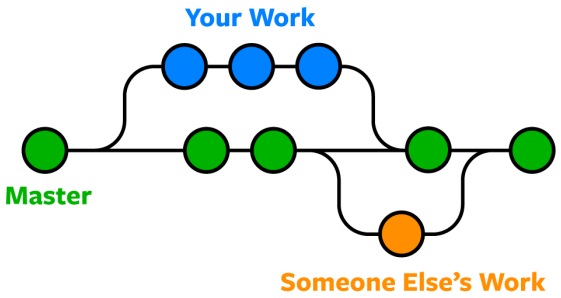
22-4-2021 **Array Handling**

T.b.d.

Git introductie

Over git

- Initial release in 2005
- Version control software
  - Creates snapshots ('pictures') of your code at a given state
- De-centralized system
  - Each working directory (user or system) owns entire 'repository', which includes code, version and tracking history



* https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

Over git

- In its core, Git is a **set** of **command line utility programs**
- Many dedicated programs exist to host Git repositories
  - GitHub
  - Bitbucket
  - GitLab
- Most development environment incorporate Git for easy version control
  - Visual Studio (Code)
  - PyCharm
  - Atom

GitHub

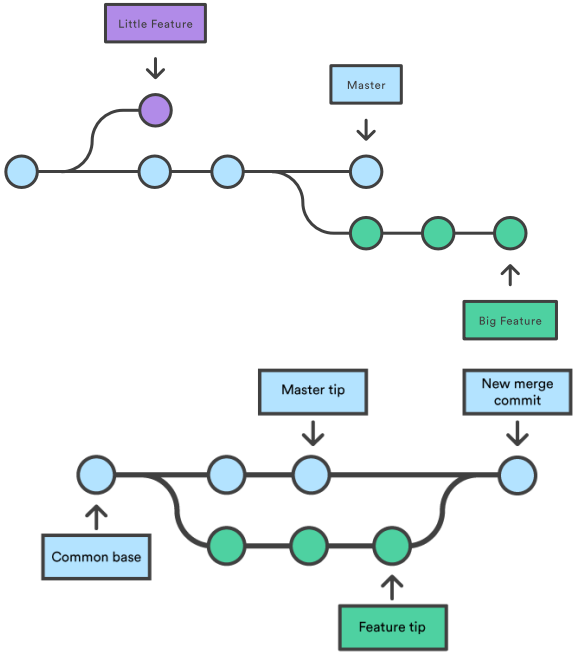
- https://github.com/
- Most popular Git interface client
- Allows users to create and share repositories and build a digital portfolio

Top Git GUI Clients For Users to Choose in 2021

- **Github Desktop**.
- **SourceTree**.
- **GitKraken**.
- **SmartGit**.
- **Git Cola**.
- GitForce.
- Giggle.
- **Magit**.

Git(Hub) terminologie

- **Branch:** a separate working subtree from the main branch
  - Can be used for features, bug fixes, etc.
- **Merge:** when a branch has been tested and approved, it can be merged back into the main branch
- **Pull request:** often follows as a result of a merge request
  - You basically ask an internal reviewer to 'review' your changes.
  - Once changes are approved, the branch gets merged into the main branch
- **Issue:** a ticket which can be created by either a user or a collaborator (depending on the repo settings) to address an issue with the code
- **Fork:** 'copy' someone else their repository and use it as your own



Git(Hub) commands

Remote = repository host (e.g. GitHub)

Local = ‘your computer’

- **Clone:** clone repository onto local machine via either HTTP or SSH
- **Add:** stage all or certain file changes that are ready to be committed
- **Commit:** commit staged snapshot with a commit message (obligated)
- **Push:** upload current branch to remote along with commits
- **Pull:** fetch specified remote’s copy of current branch and merge into local copy
- **Add:** stage all changes in directory for the next commit
- **Diff:** shows difference between working directory and last commit of the current branch

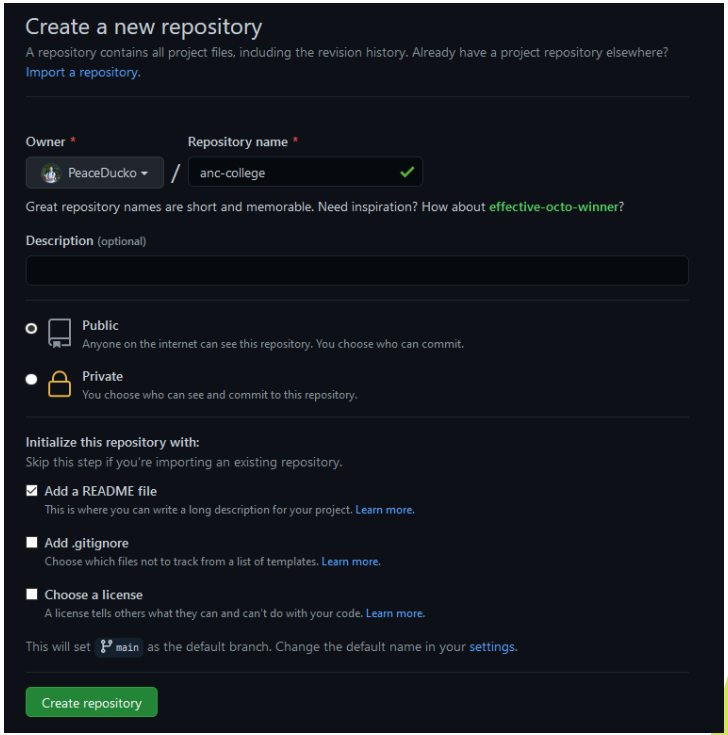
Waarom Git gebruiken voor projecten

- Superior version control system
- Both big and small companies use Git as their software VCS
- Branches are safeguards to prevent accidentally uploading code that breaks your production application

Git in de praktijk

Repository aanmaken

- **Always** add a README file
  - The contents of this file are visible on the homepage of your repository and should show people what your repository is about
- Public or private repository depends on your needs
  - Public is best if you want to share your creations with anyone else
  - Private is more suited for organizations or when working with sensitive content



Repository aanmaken

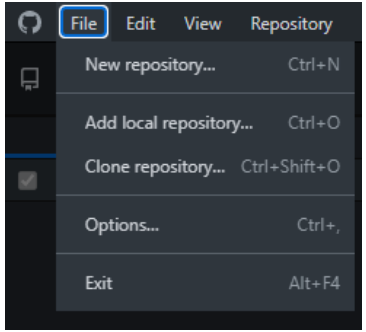
- **Gitignore** is essential!
  - In this file you specify which file you don't want to keep local (thus not upload remotely to GitHub).
  - Git will check if a file or folder is present in the .gitignore and will then skip it in the *push*
  - Essential when working with e.g. public/private key pairs or a credentials file
  - You can optionally set a .gitignore template (e.g. Python) to let Git automatically ignore certain unnecessary Python library files

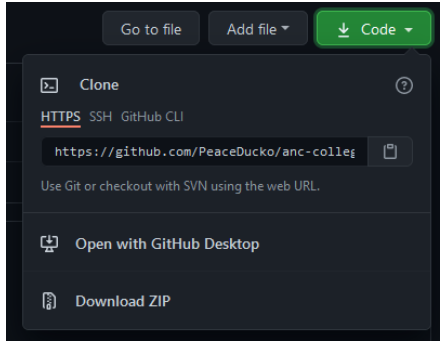
129 lines (105 sloc) | 1.76 KB

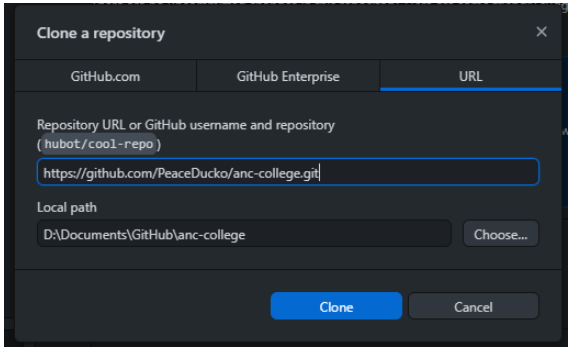
```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
pip-wheel-metadata/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST
```

Bestaande repository clonen

- Either GitHub Desktop or using the CLI (command line interface)
- In GitHub Desktop: File > Clone repository
- Choose either to:
  - Clone directly from GitHub.com (easy)
  - Clone from URL (more flexible)
- When cloning using URL, copy HTTPS link from the code repository you want to clone
- Define local path to store repository

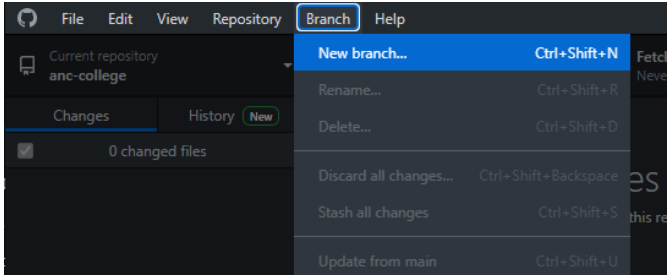


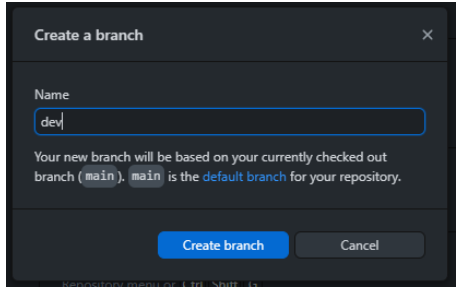


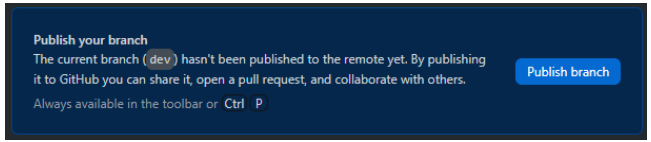


Nieuwe branch aanmaken

- Git will create a ‘main’ branch by default
  - This branch is linked to production code most of the time
  - We don’t want to accidentally push code that is not working to the main branch
  - So…we create a new branch!
- Create new ‘dev’ (development) branch in either GitHub Desktop or on GitHub
  - You will see that this branch is based on our current ‘main’ branch, meaning that this branch will merge with the *main* branch after a pull or merge request
- Publish the branch
  - On GitHub Desktop, the new branch will be local, thus we have to upload it to the remote repository





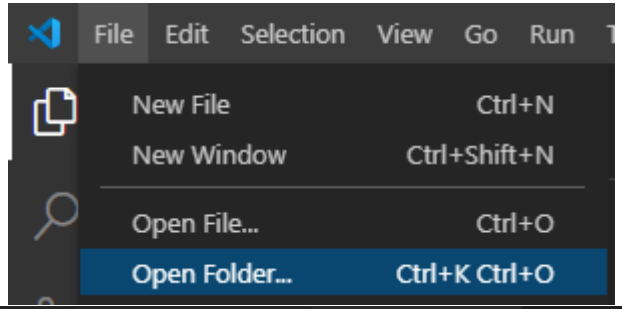


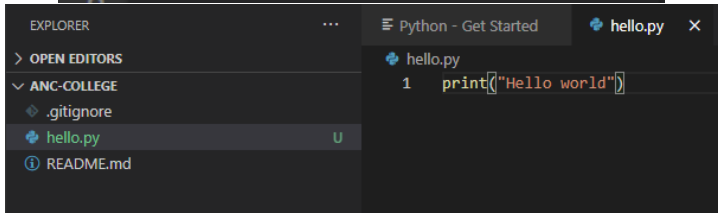
Code toevoegen aan de repository

- Let's check if we have everything:
  - ✓ Created remote repository
  - ✓ Cloned our repository
  - ✓ Created a standalone branch
  - ✓ Published the branch
  - Uploaded code to our new branch
  - Merged our current branch with the main branch

Code toevoegen aan de repository

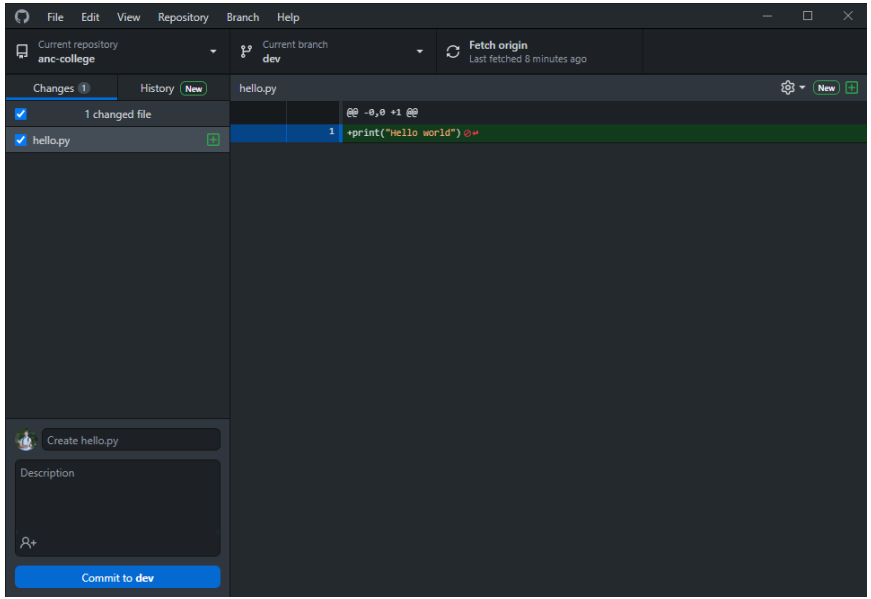
- Open your favorite IDE (mine is Visual Studio Code)
- Open the folder which houses your local repository
  - Remember, this is the path which we defined when cloning our repository
- Add a new (random) file
  - *Bonus points for originality*
- In VSCode (and some other IDEs as well), you will see that the name of the new file is green
  - We know that every user who clones a repository owns the entire information about that repository
  - Meaning that IDEs like VSCode can read whether this file we created is new





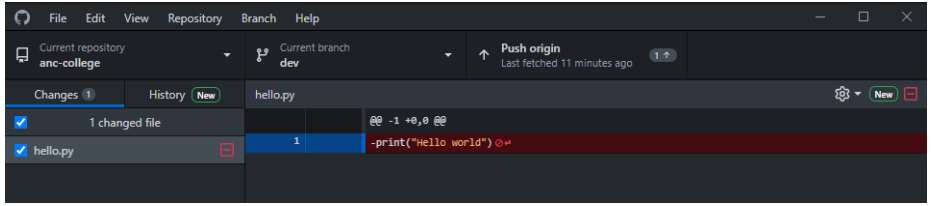
Code toevoegen aan de repository

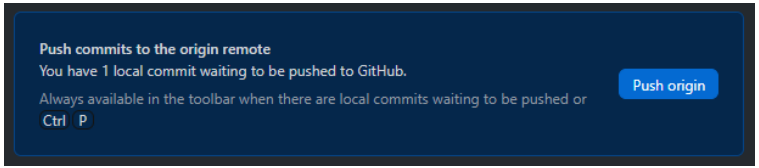
- After saving the new file, head back to GitHub Desktop
  - You will see that GitHub has detected a file change, and has already added a commit message for you as well (in the CLI, you would have to specify a message yourself)

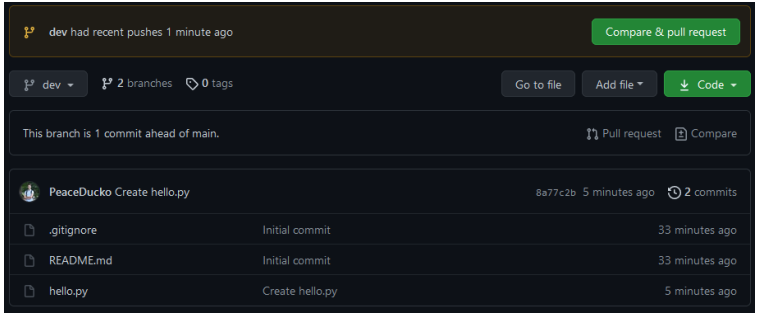


Code toevoegen aan de repository

- You can commit as many changes as you want without pushing them to the remote GitHub repository
- We could, for example, realize that we did not need *hello.py* after all, and delete it from our directory
    - Then we head back to GitHub Desktop and repeat the cycle. Check changes > Commit
- Let's in this case assume that we want to push *hello.py* to our remote repository
- You are asked if you want to 'Push commits to the origin remote'
- If we push our changes, they are uploaded to our remote GitHub repository





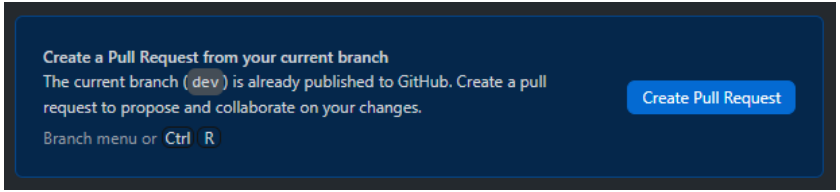


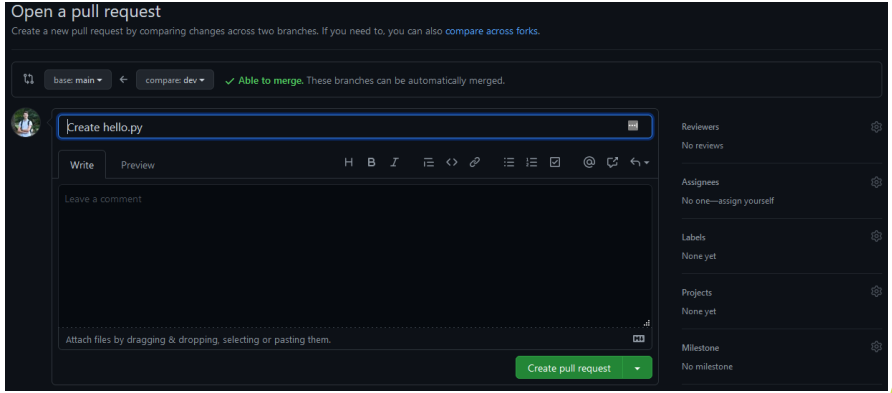
Huidige branch mergen

- Let's check if we have everything:
  - ✓ Created remote repository
  - ✓ Cloned our repository
  - ✓ Created a standalone branch
  - ✓ Published the branch
  - ✓ Uploaded code to our new branch
  - • Merged our current branch with the main branch

Huidige branch mergen

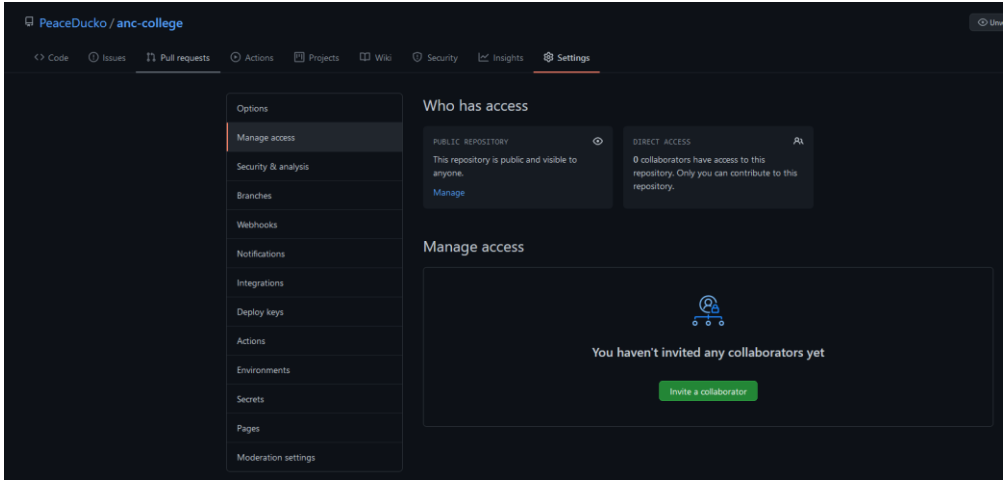
- After we pushed our changes to the *dev* branch, we can create a Pull Request to merge this branch with our *main* branch
- When clicking on 'Create Pull Request', you will be guided to the GitHub site where you can create your PR
- Organizations often have checks installed to make sure that no PR is accidentally merged with the main branch
    - For example: adding one or more reviewers who must approve the PR first before it gets merged

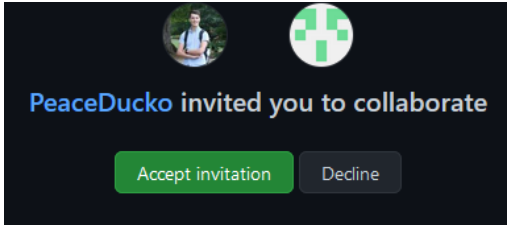




Huidige branch mergen

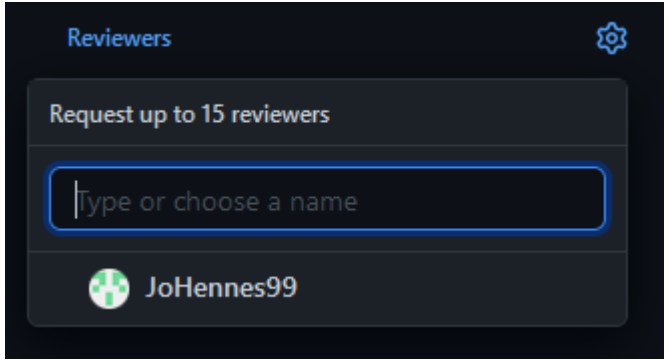
- **Collaborators** are people within your group or organization who can commit, push, create branches, review pull requests and much more
- Option under ‘Settings’ > ‘Manage access’
- Invite collaborator by username, full name or email
- New collaborator will receive invitation mail which they can accept or decline





Huidige branch mergen

- During or after creating our PR we can add reviewers
- Reviewer(s) receive an email asking them to review the PR



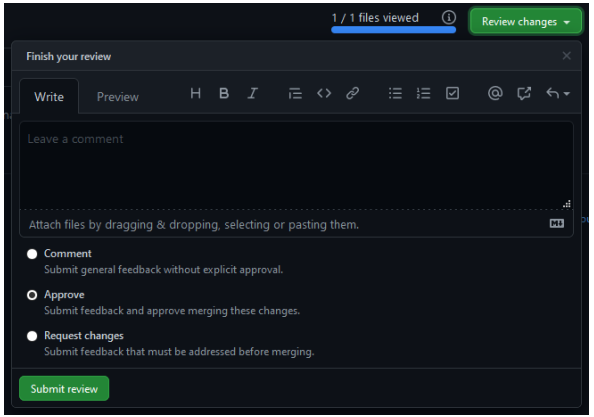
@PeaceDucko requested your review on: #1 Create hello.py.

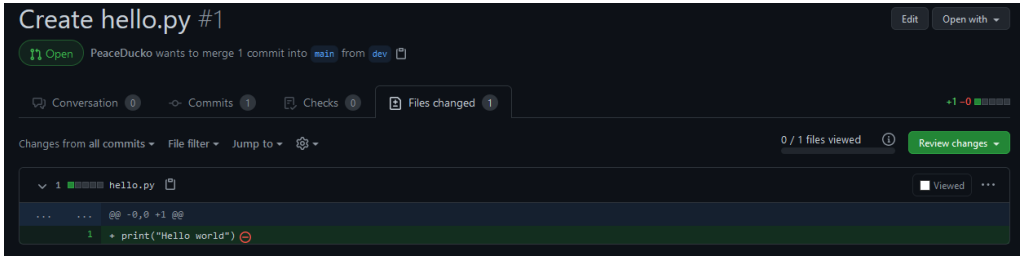
—

You are receiving this because your review was requested.
Reply to this email directly, view it on GitHub, or unsubscribe.

Huidige branch mergen

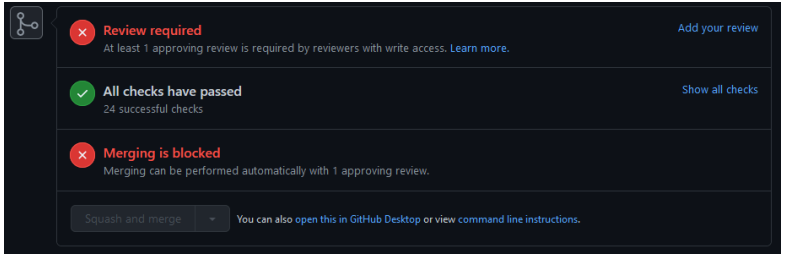
- After clicking on ‘Add review’, you can see the changes the author of the PR made to the existing branch
- You can view the changes of each file and checkmark them so the author knows you reviewed them
- Once you’re done you can either
  - Comment on the code without approval
  - Approve the code
  - Request changes the author must apply to the code before it can be accepted

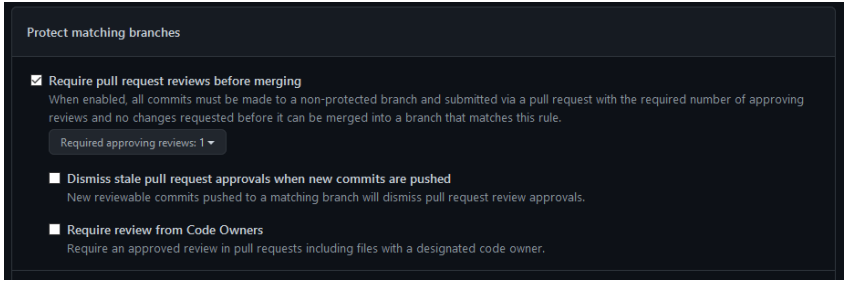




Huidige branch mergen

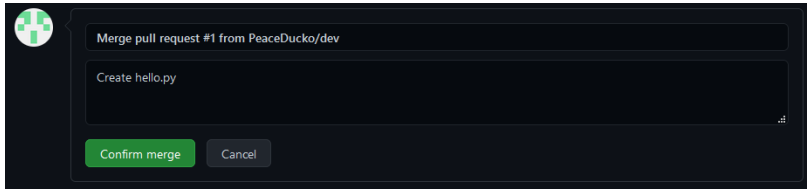
- Because we don't have strict checks built in, the author of the PR could merge the PR without any review
  - In organizations, this is **not** the case! (example from association)
  - Under the 'Branches' option in the repository settings, you can require PR reviews before merging and set the number of reviewers required

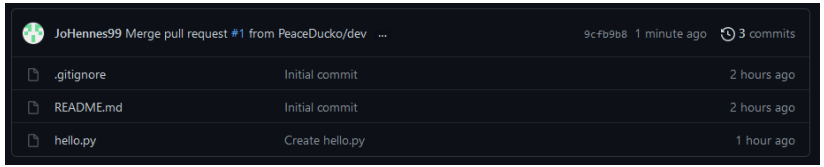


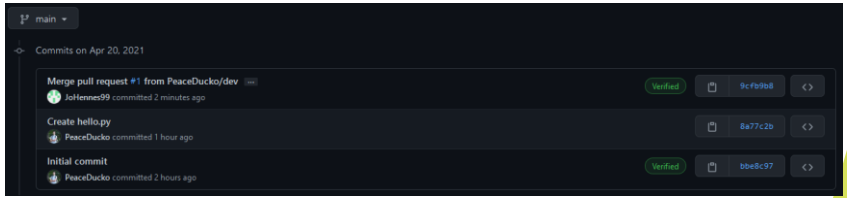


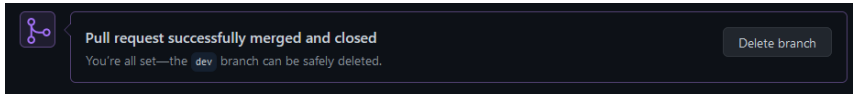
Huidige branch mergen

- After all checks are done you can use the ‘Merge pull request’ button to merge the working branch with the base branch!
  - Checks can include code testing, linting (formatting), deploy to server, etc.
- Both the author as well as the reviewer(s) can merge the PR
- PR will first be turned into a commit from the current branch into the base branch
- Feature or bug-only branches are recommended to be deleted. However, it is useful to keep more abstract branches like *development* or *testing*









Adempauze

Git is groot

- Practice. Practice. Practice.
- The Git infrastructure is massive, you won't master it in a day…or a week…or a year
- Learn on-the-go by version controlling your projects
- Master Git by sticking to the best practices
  - Branch new developments
  - Review your changes
  - Provide clear commit messages

Probeer het zelf

Probeer het zelf

1) Maak een account aan up GitHub: https://github.com/join
2) Download GitHub Desktop: https://desktop.github.com/
3) Stuur je GitHub gebruikersnaam in de Teams chat
    a) Ik voeg je toe als collaborator
    b) Houd je mail in de gaten voor een uitnodiging om deel te nemen aan de repo
4) Maak een nieuwe branch aan volgens het volgende format: <voorletter achternaam>
    a) In mijn geval: sram
    b) Zorg dat je je in de nieuwe branch bevindt! Anders kan je niet committen
5) Open je favoriete IDE en voeg een bestand met inhoud toe (bijv. een .py bestand)
    a) Zorg dat de bestandsnaam uniek is t.o.v. je medestudenten, anders krijgen we later een merge conflict
6) Commit jouw changes met een bijpassende message en push deze naar jouw branch
7) Maak een Pull Request aan en voeg (optioneel) één van je medestudenten toe als reviewer
8) Review en approve één Pull Request van je medestudent
9) Merge de Pull Request naar de *main* branch

Git in de CLI

Git in de CLI

- Using the GitHub Desktop application (or even Sourcetree for example when working with Bitbucket) is extremely convenient
  - Easy to set up
  - Fast
  - Straightforward
- …but sometimes we want a more flexible approach
- Introducing the Git CLI (command line interface)
- The Git CLI has the same functionalities as we have already seen in the GitHub Desktop demo
- The CLI does not need a graphical user interface (GUI) to be used

- Download Git CLI: https://git-scm.com/downloads

Git in de CLI

- Your commands are the same
  - Except they're not presented in a graphical fashion

- *git clone <http link>*
  - Clone a repository into your current working directory
- *git commit –m <commit message>*
  - Commit changes in your working directory
- *git push*
  - Push local committed changes to the remote repository
- *git pull*
  - Pull remote changes into local directory

```
PS C:\Users\srram> git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status
```

Vragen?

hogeschool Windesheim

Radboud University