Dive in to Git and GitHub

using Zoom - Virtual Classroom Edition

https://compicampus-git-intro.website

CompiCampus

2020-10-07

Roman Plessl

Agenda and Goals for Today

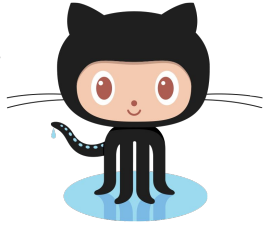
1. Git - Introduction to Version Control Systems, History of Git
2. Learning Path and Exercises based on your knowledge and experience with Git / GitHub / VCS
3. A lot of practical Exercises:
    a. Basic Git Workflow, Basic Git Commands
    b. Create a GitHub Account, create new Repositories
    c. How to use GitHub, How to use local Git with GitHub
    d. Working with Git branches
    e. Interact with each other's and other coders:
       Git Branching, Code Changes and Commits, Pull Requests, Merging, Comments
4. Host your personal, organization, and project sites with GitHub Pages.
5. How to use GitHub (II): Issues Tracking and Feature Requests, Task Management and Wiki, Insights and Settings

Git

- **Git** is a distributed version-control system for tracking changes in source code.
- **Git** was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development.
- **Git** is maintained since 2005 by @gitster - Junio C Hamano.
- Over 70% of all Code Projects use **Git**.
- **Git** is now on Version 2.28

GitHub

- **GitHub** is a web-based hosting service for version control using Git. It is mostly used for computer code.
- **GitHub** offers all of the distributed version control and source code management functionality of Git as well as adding its own features (continuous integration (CI) using GitHub Actions, bug tracking, feature requests, task management, and wikis for every project)
- **GitHub** was bought 2018 by Microsoft (for 6.4 Mrd €).



Git

- **Git** is a distributed version-control system for tracking changes in source code.
- **Git** was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development.
- **Git** is maintained since 2005 by @gitster - Junio C Hamano.
- Over 70% of all Code Projects use **Git**.
- **Git** is now on Version 2.28

GitHub vs. GitLab

- **GitHub** is a web-based hosting service for version control using Git. It is mostly used for computer code.
- **GitHub** offers all of the distributed version control and source code management functionality of Git as well as adding its own features (continuous integration (CI) using GitHub Actions, bug tracking, feature requests, task management, and wikis for every project)
- **GitHub** was bought 2018 by Microsoft (for 6.4 Mrd €).
- *site note:* **GitLab** has very similar functionality and can be run also selfhosted (as it is done at ETH)

History about Git
Motivation for Git

History about Git

Git started in 2005 by Linus Torvalds (Linux Inventor)

- to aid the Linux Kernel development, and
- to help the developer to manage their code change patches (till 2002), and
- as a replacement for the proprietary and “free-of-charge” software BitKeeper (till 2005).

Linus Torvalds speaking at the LinuxCon Europe 2014 in Düsseldorf

(Picture by Krd, CC BY-SA 4.0)

History about Git

Design Goals for Git:

- Speed
- Simple design
- Strong support for non-linear development
  (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently
  (speed and data size)

History about Git

Focus on Scale

Daten und Zahlen zu den jüngsten Versionen des Linux-Kernels

Kernel-Version	Anzahl Dateien[1]	Zeilen Quelltext (Ohne Doku)[2]	Entwick-lungs-zeitraum	Commits (Ohne Merges)[3]	Diffstat[4]
Linux 4.18	61.003	25.280.872 (23.183.236)	70 Tage	14.432 (13.283)	13.141 files changed, 583.336 insertions(+), 682.028 deletions(-)
Linux 4.19	61.734	25.588.455 (23.449.221)	70 Tage	15.204 (14.043)	11,693 files changed, 552.223 insertions(+), 244.235 deletions(-)
Linux 4.20	62.481	25.955.520 (23.776.585)	63 Tage	14.995 (13.844)	11402 files changed, 685.027 insertions(+), 317.959 deletions(-)
Linux 5.0	63.135	26.203.035 (23.933.016)	70 Tage	13.921 (12.808)	12.100 files changed, 579.084 insertions(+), 331.570 deletions(-)
Linux 5.1	63.873	26.459.776 (24.141.004)	63 Tage	14.160 (13.034)	11.977 files changed, 545.423 insertions(+), 288.683 deletions(-)
Linux 5.2	64.587	26.552.127 (24.175.296)	63 Tage	15.089 (14.024)	30.888 files changed, 624.857 insertions(+), 532.510 deletions(-)
Linux 5.3	65.261	27.141.312 (24.708.822)	70 Tage	15.784 (14.605)	13.983 files changed, 1.189.832insertions(+), 600.665 deletions(-)

[1] git ls-tree -r --name-only HEAD | wc -l
[2] find . -type f -not -regex '\./\.git/.*' | xargs cat | wc -l; echo "($(find . -name *.[hcS] -not -regex '\./\.git/.*' | xargs cat | wc -l))"
[3] git-log --pretty=oneline v*x*.(*y-1*)..v*x*.(*y*) | wc -l; echo "($(git-log --pretty=oneline --no-merges vx.(y-1)..vx.(y) | wc -l))"
[4] git diff --shortstat v*x*.(*y-1*)..v*x*.(*y*)

Facts and Figures on the amount of Linux Kernel Changes
(creator Thorsten Leemhuis, heise online)

Based on your level with Git / GitHub:

My Idea of a Learning-Path:

1. Touch and see how Git can be used locally
2. New Repositories can be done by hand and using GitHub in a Basic way
3. Afterwards A. How to use Git branches (incl. Deepening in this topic)
   B. GitHub (including automated and automagically steps)

Beginners in Git and GitHub should start with the one after next slide:

- Introduction to Version Control System (p. 11) and Intro to Git (p. 13)
- Install Git (p. 14), use Git locally (p. 17) and
- then with the Basic Setup of GitHub (p. 31).

Intermediate and Advanced users, see next slide.

Based on your level with Git / GitHub:

Intermediate Git and GitHub Users should start with:

- Understanding Git Cheat-Sheet (p. 28)
- GitHub Setup and local-remote Exercises (p. 32, p. 37ff), afterwards the
- Git Branching, Merging and Rebase Exercise and Riddles (p. 44)

More Intermediate Git and GitHub Users should start with:

- Understanding Git Cheat-Sheet (p. 28) and Git Reset Mechanism (p. 30)
- Git Branching, Merging and Rebase Exercise and Riddles (p. 44)
- GitHub Setup and local-remote Exercises (p. 32, p. 37ff), afterwards the
- Collaborative Working (p. 46)

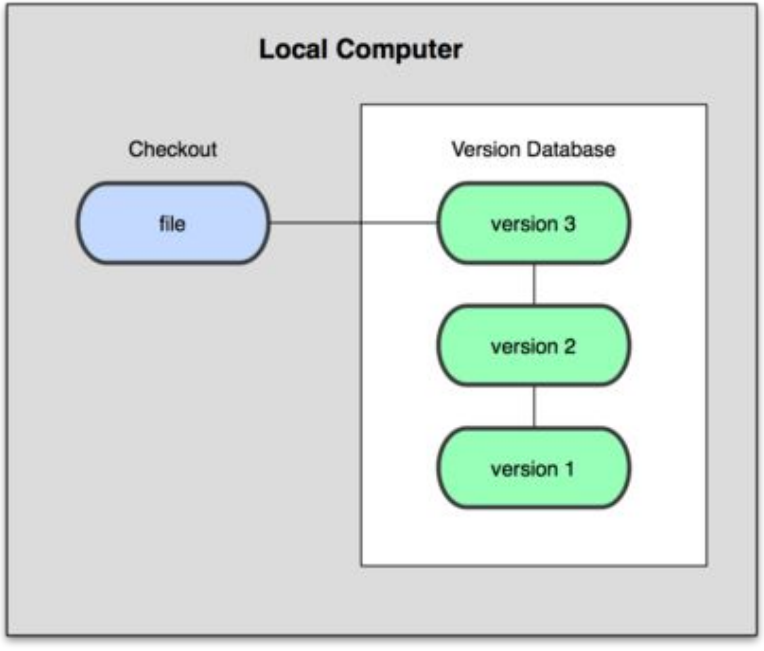
- Setup a Website with GitHub Pages and use Issue Tracker et al of GitHub afterwards (p. 49)

Introduction to Git and Version Control System

What is a version control system?

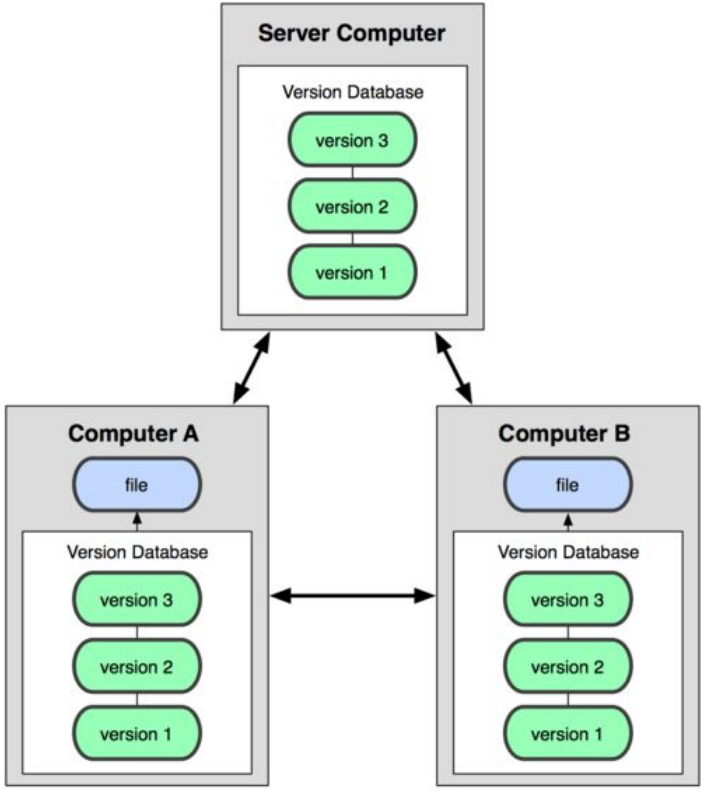
Version control system (VCS) in general

- A system that keeps records of your changes
- Allow for collaborative development and exchange of content or code
- Allows you to know who made what change and when
- **Allows you to revert any changes and go back to previous state**



What is distributed version control?

- Distributed version control
- Users keep entire code and history on their location machines
- Users can make any changes without internet access
- (Except pushing and pulling changes from a remote server)
- (the internet access world was different 15 years ago)



Install Git locally

If necessary: Install Git locally
(check if already installed with `git status`)

mac OS (preinstalled, check with `git status`):

- Otherwise: https://git-scm.com/download/mac

Windows (sometimes preinstalled, check with `git status`):

- Install https://git-scm.com/download/win
- or use WSL 2 https://docs.microsoft.com/en-us/windows/wsl/install-win10

Linux:

- Debian / Ubuntu: `sudo apt install git`
- Fedora/RHEL: `sudo dnf install git`
- Fedora / RHEL (older): `sudo yum install git`

Use Git at ETH (Windows)

- Either open a command prompt (cmd) and check if the 'git' command is part of known commands (PATH):

  ```
  git status
  ```

  or if that is not working:

- Open in the File Explorer

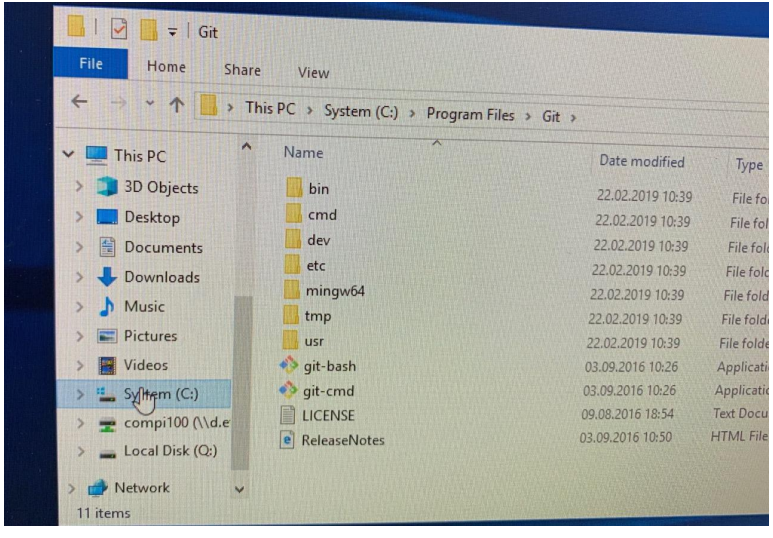
  This PC → System (C:) → Program Files → Git

  and start

  ```
  git-bash
  ```

  or

  ```
  git-cmd
  ```



Basic Git Workflow

Basic Git Workflow (I)

Short overview in Basic Git Workflow, the practical exercises will follow in the next chapter

Tell Git to track **that folder** that houses your project files.

```
cd <path-to-project-folder>
```

Use the following command to convert your project folder into a local git repository:

```
git init
```

(Important: do **not execute** git init more than once in a project folder.)

Basic Git Workflow (II)

You can tell Git to mark certain files for saving. The git add command is responsible for this. At the end of the command, enter the path to the files you want to mark (relative to the project folder), separating each file's relative path with a space.

The command below tells git to **MARK** 2 files for saving (changed_file_1 and changed_file_2 in the subfolder named folder1):

```
git add changed_file_1 folder1/changed_file_2
```

You can also tell Git to **MARK ALL CHANGED/EDITED** files for saving. The code below does just that:

```
git add .
```

There are times when you want to use the `git add .` command to MARK ALL edited files. **Actually, most times that's what you should do.** But there may be some files and folders that have no business whatsoever in a git repository.

Basic Git Workflow (III)

To finally save the **MARKED** files to the local repository, enter the following code in your terminal / command prompt:

```
git commit -m "short descriptive message"
```

If you want to exclude files, create a file on the top level with the name **".gitignore"**, with the file names to exclude.

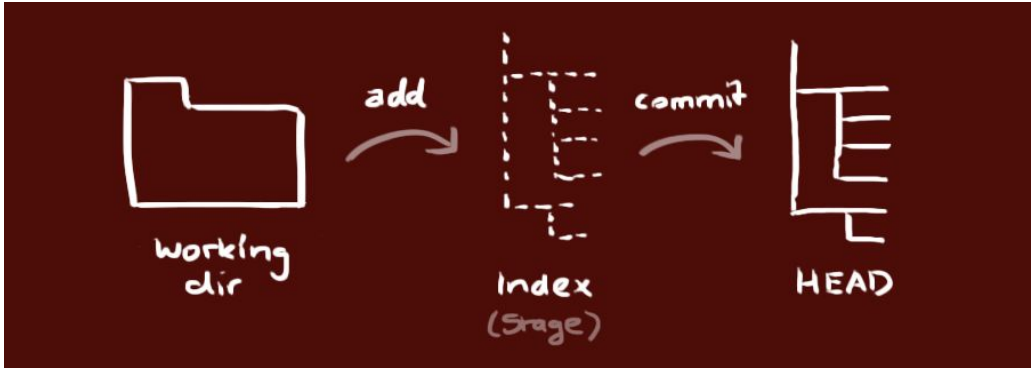
For further detail look at these examples and helping tool:

- https://help.github.com/en/articles/ignoring-files
- https://www.gitignore.io

Basic Git Workflow (IV)

Your local repository consists of three "trees" maintained by git.

- the first one is your **`Working Directory`** which holds the actual files.
- the second one is the **`Index`** which acts as a staging area
- and finally the **`HEAD`** which points to the last commit you've made.



Basic Git Commands

Basic Git Commands

- `git init` initializes a brand new Git repository and begins tracking an existing directory.

  It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.

- `git clone` creates a local copy of a project that already exists remotely.

  The clone includes all the project's files, history, and branches.

- `git add` stages a change.

  Git tracks changes to a developer's codebase, but it's necessary to stage and take a snapshot of the changes to include them in the project's history. This command performs staging, the first part of that two-step process. Any changes that are staged will become a part of the next snapshot and a part of the project's history. Staging and committing separately gives developers complete control over the history of their project without changing how they code and work.

Basic Git Commands

- `git commit` saves the snapshot to the project history and completes the change-tracking process.

  In short, a commit functions like taking a photo. Anything that's been staged with git add will become a part of the snapshot with git commit.

- `git status` shows the status of changes as untracked, modified, or staged.

- `git branch` shows the branches being worked on locally.

- `git merge` merges lines of development together.

  This command is typically used to combine changes made on two distinct branches. For example, a developer would merge when they want to combine changes from a feature branch into the master branch for deployment.

Basic Git Commands

- `git pull` updates the local line of development with updates from its remote counterpart. Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.

- `git push` updates the remote repository with any commits made locally to a branch.

Basic Git Commands (20 Min)

For practical examples of these commands have a look and try these commands:

https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Global Settings for locally installed Git (5 Min)

Really recommended step after installation of local installed Git:

After the installation of Git, open terminal (if you're using Linux or Mac) or command prompt (if using Windows). Please keep your terminal open, we'll be using it for a while in the next exercises.

Now let's tell git things about you—your name and email really. Git needs those info so it can confidently tie your identity to any commits you make; and then when someone comes asking, *who made those changes?*, Git can say, it was this guy or that lady.

Punch the following **TWO** instructions into your terminal/command prompt.
You would only have to do it this one time:

```
git config --global user.name "Your Name Comes Here"
git config --global user.email "you@yourdomain.example.com" (same as on GitHub)
```

Git Cheat Sheet

https://services.github.com/on-demand/resources/cheatsheets/

GitHub
GIT CHEAT SHEET
V1.0.1

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

INSTALL GIT
GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Git for All Platforms
http://git-scm.com

CONFIGURE TOOLING
Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```
Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```
Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```
Enables helpful colorization of command line output

CREATE REPOSITORIES
Start a new repository or obtain one from an existing URL

```
$ git init [project-name]
```
Creates a new local repository with the specified name

```
$ git clone [url]
```
Downloads a project and its entire version history

MAKE CHANGES
Review edits and craft a commit transaction

```
$ git status
```
Lists all new or modified files to be committed

```
$ git diff
```
Shows file differences not yet staged

```
$ git add [file]
```
Snapshots the file in preparation for versioning

```
$ git diff --staged
```
Shows file differences between staging and the last file version

```
$ git reset [file]
```
Unstages the file, but preserve its contents

```
$ git commit -m "[descriptive message]"
```
Records file snapshots permanently in version history

GROUP CHANGES
Name a series of commits and combine completed efforts

```
$ git branch
```
Lists all local branches in the current repository

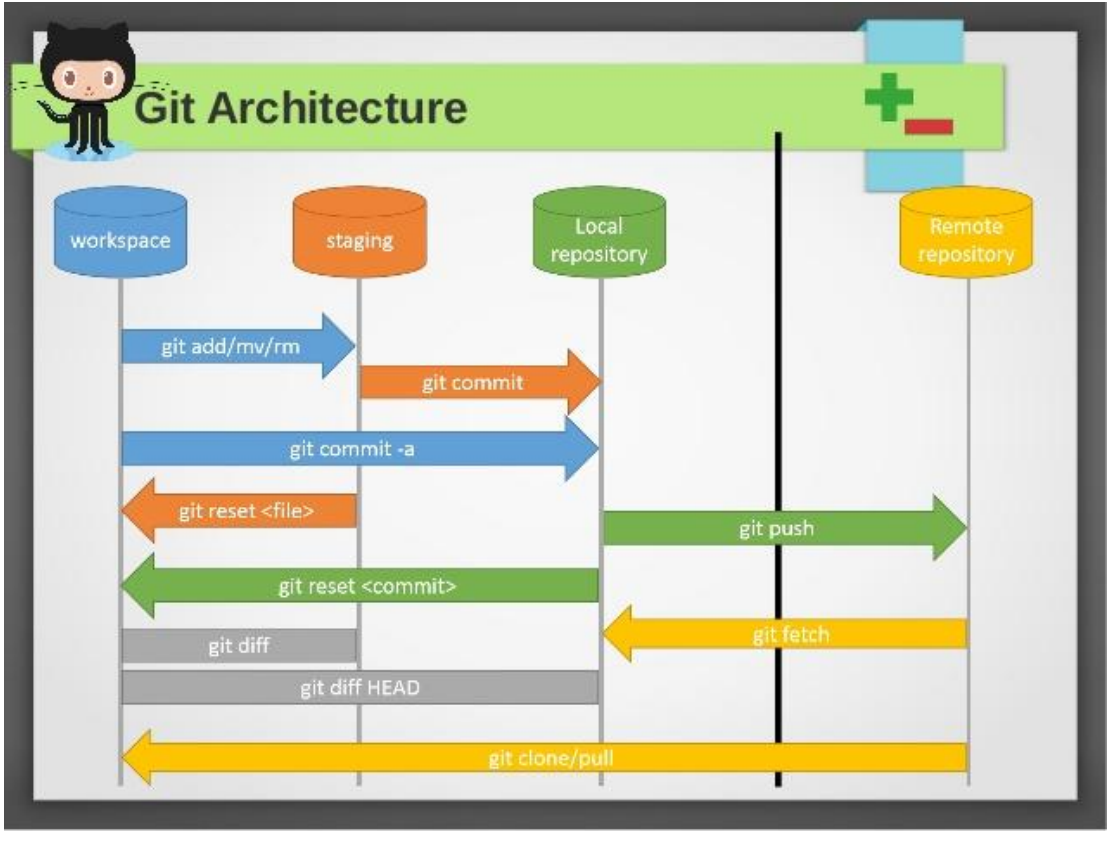
```
$ git branch [branch-name]
```
Creates a new branch

```
$ git checkout [branch-name]
```
Switches to the specified branch and updates the working directory

```
$ git merge [branch]
```
Combines the specified branch's history into the current branch

```
$ git branch -d [branch-name]
```
Deletes the specified branch

Git Architecture



Advanced: How to “git reset” (spare time)

A good explanation of how to reset a checkout to a previous version is described here:

https://www.atlassian.com/git/tutorials/undoing-changes/git-reset

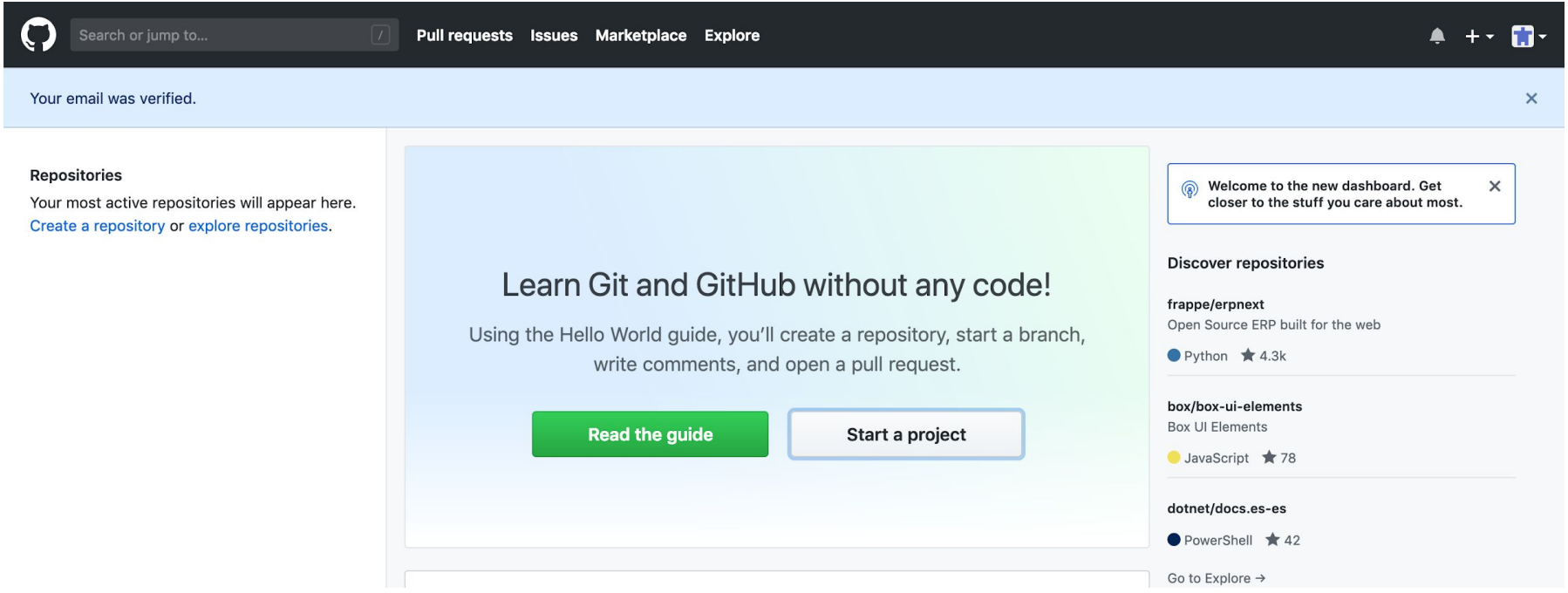
How to use GitHub (I)

How to use GitHub (I): Create a GitHub Account, create new repositories

If you haven't already a github account create one:

- Login to https://github.com/join
- Choose a not already taken username for the account and fill in your personal data in “Create your personal account”. To complete, solve the riddle.
- “Choose your plan”: Select the “free” plan
- Enter your experience in “Tailor your experience”
- Verify your Email Address in your Email Account.

Create your first “hello-world” example (I)



Create your first “hello-world” example (II)

- Open the “Read the Guide” Website … :
  https://guides.github.com/activities/hello-world/
- … AND Open the “Start a project” Website
  https://github.com/new
- follow the “hello world” example of GitHub,

  but name the repository “`compicampus-git-intro`” instead of “hello world”

Create your first “hello-world” example (III)

You should have

- edited your README.md file and enhanced the content with Text
- created a new branch
- created a pull request
- merged the branch to the master branch

Enhance your README.md with Markdown

- Open your README.md in your “compicampus-git-intro” Repository and enhance this with more Markdown
  - https://guides.github.com/features/mastering-markdown/
  - https://github.com/ikatyang/emoji-cheat-sheet/blob/master/README.md
- Build Documentation parts with
  - Code
  - Emojis
  - Tables

How to use local Git with GitHub

How to use local Git with GitHub (I): Cloning using HTTPS

- Create a new Repository on GitHub including your GitHub Username "`compicampus-USERNAME`" (e.g. `compicampus-rplessl`)
- New user / re-open the shell or prompt with the git command line interface
- Clone your new GitHub Repository locally with HTTPS as documented here: https://help.github.com/en/articles/cloning-a-repository
- Delete the Repository on GitHub as documented here: https://help.github.com/en/articles/deleting-a-repository
- Delete your local checkout

Global Settings for locally installed Git
(if not already done before)

Really recommended step after installation of local installed Git:

After the installation of Git, open terminal (if you're using Linux or Mac) or command prompt (if using Windows). Please keep your terminal open, we'll be using it for a while in the next exercises.

Now let's tell git things about you—your name and email really. Git needs those info so it can confidently tie your identity to any commits you make; and then when someone comes asking, *who made those changes?*, Git can say, it was this guy or that lady.

Punch the following **TWO** instructions into your terminal/command prompt.
You would only have to do it this one time:

```
git config --global user.name "Your Name Comes Here"
git config --global user.email "you@yourdomain.example.com" (same as on GitHub)
```

How to use local Git with GitHub (II): Cloning using HTTPS

Exercise: Contribute to an existing repository

- Create a new Repository on GitHub including your GitHub Username "compicampus-USERNAME" (e.g. compicampus-rplessl)
- **Contribute to an existing repository**

```
# download a repository on GitHub.com to our machine
git clone https://github.com/<USERNAME>/compicampus-USERNAME.git

# change into the `repo` directory
cd repo

# create a new branch to store any new changes
git branch my-branch
```

How to use local Git with GitHub (III): Cloning using HTTPS

Exercise: Contribute to an existing repository

```
# switch to that branch (line of development)
git checkout my-branch

# make changes, for example, edit `file1.md` and `file2.md` using the
# text editor (like visual code, nano, vim,… )

# stage the changed files
git add file1.md file2.md

# take a snapshot of the staging area (anything that's been added)
git commit -m "my snapshot"

# push changes to github
git push --set-upstream origin my-branch
```

How to use local Git with GitHub (IV): Push Content using HTTPS

Exercise: Start a new repository and publish it to GitHub

- **Start a new repository and publish it to GitHub**

  First, you will need to create a new repository on GitHub (by clicking the Website).

  You can learn how to create a new repository in our Hello World guide. Do not initialize the repository with a README, .gitignore or License. This empty repository will await your code.

```
# create a new directory, and initialize it with git-specific functions
git init my-repo

# change into the `my-repo` directory
cd my-repo

# create the first file in the project
touch README.md
```

How to use local Git with GitHub (V): Initialisation using HTTPS

Exercise: Start a new repository and publish it to GitHub

```
# git isn't aware of the file, stage it
git add README.md

# take a snapshot of the staging area
git commit -m "add README to initial commit"

# provide the path for the repository you created on github
git remote add origin https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git

# push changes to github
git push --set-upstream origin master
```

Git is operating on branches

Working with Git branches

After the basic from Git / GitHub we now learn how git is meant to be used: using Git branches and making changes in the project safely off to one side, and merging them back into the original project (master) once they have been proved to be correct.

A very good online tutorial on Git Branches and how they behave is here

- https://learngitbranching.js.org/?locale=en_US

Interact with each other and other coders

Interact with each others and other coders

Forking: After using Git and GitHub by yourself for a while, you may find yourself wanting to contribute to someone else's project. This process is known as ***forking***.

- Follow this tutorial with the additional informations below:
  https://guides.github.com/activities/forking/

- Clone the forked repository to your local machine:
  `git clone https://github.com/<username>/Spoon-Knife.git`

- Open the Visual Code Editor on your machine (or whatever your prefered Editor is)
- Change files locally
- `git commit` and `git push` them to your Repository on Github

- Create a compare and pull request as described in the Guide

Interact with each others and other coders

Pull Requests, Comments on Pull Requests, Merging:
Now we are working together on our own repositories and work together

1. Build Groups of 2
2. Ask your colleague for his github username and for a repository-name
3. Fork the Repository of your colleague on GitHub
4. Clone it to your local machine
5. Change some code and text parts in the Repository
6. Commit your changes
7. Push it to GitHub back
8. Create a pull request (https://help.github.com/en/articles/creating-a-pull-request)
9. And comment parts of the code changes in the pull request to you (https://help.github.com/en/articles/commenting-on-a-pull-request)

GitHub Pages
Host your static website

GitHub Page Website

GitHub Pages are public webpages hosted and easily published through GitHub

- GitHub Pages are free of charge static websites
- The GitHub Page Website is build with the Jekyll Framework
- Websites are themeable
- Content and Style can be modified remotely via the GitHub Website or locally in the checkout on your computer
- Website Content is given in Markdown
- A custom Website Address (Domain) can be chosen (incl. SSL Certificate from Letsencrypt) https://help.github.com/en/articles/using-a-custom-domain-with-github-pages

Create your personal GitHub Page Website

- Login and go to your GitHub Dashboard

- Create a repository called '<username>.github.io'

- Add a Repository Description
- Select the 'Jekyll' Template from the .gitignore Select-Box
- Select a License (Apache License 2.0)

- And follow this guide to create your public https://<username>.github.io Website
  https://guides.github.com/features/pages/

- Additional: If you have a unused spare DNS Domain try to setup the a custom domain website:
  https://help.github.com/en/articles/using-a-custom-domain-with-github-pages
  and activate force HTTPS afterwards

Issues Tracking and Feature Requests, Task Management and Wiki, Insights and Settings

Collaborative Features included in GitHub

GitHub includes several collaborative features for handling, enhancing and reporting of the code / of the **project**.

- Issue and Feature Request Tracker

  https://github.com/USERNAME/REPOSITORY/issues

  See Guide https://guides.github.com/features/issues

- Project Board (handling 1 or more repositories)

  https://github.com/USERNAME/REPOSITORY/projects

  See Guides: https://help.github.com/en/articles/about-project-boards and https://github.com/features/project-management/

Collaborative Features included in GitHub

- Package and Release Code

  https://github.com/features#code-hosting

- All Features in one big page

  https://github.com/features#code-hosting

Further Reading and Exercises

Further Exercises and Reading

Book Recommendation:

- The complete “Pro Git” book is available at: https://git-scm.com/book/en/v2/

Especially read and test the stuff written in the branching chapter:

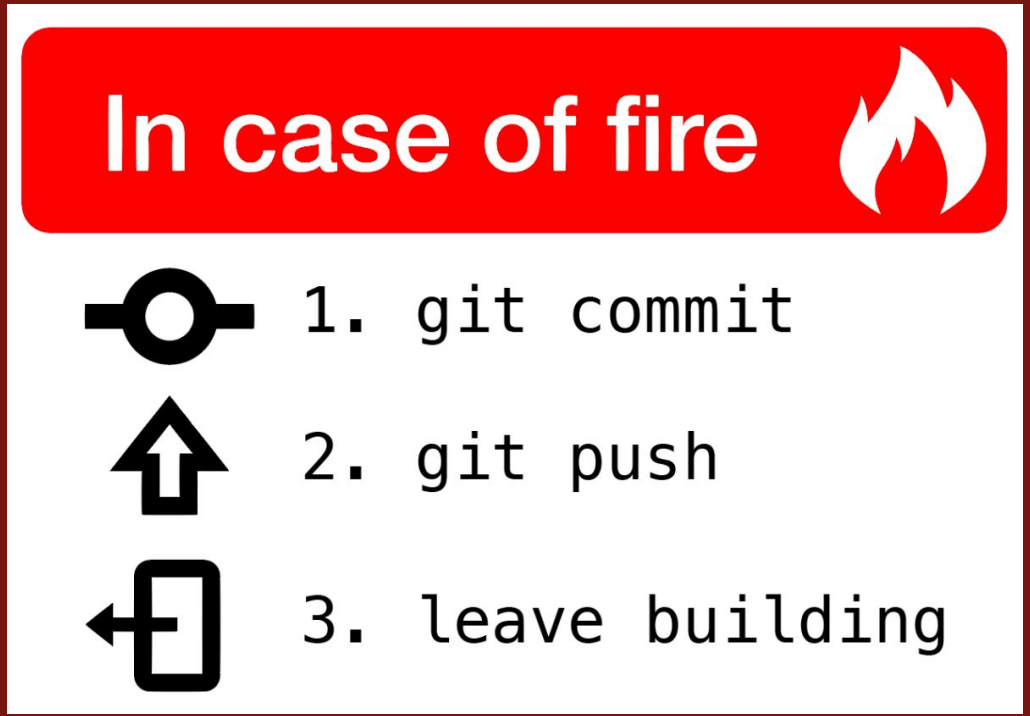
- https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Further Exercises and Reading

Further Exercises:

- Nearly always - after forking a repository, a re-synchronisation is necessary (https://help.github.com/en/articles/syncing-a-fork)

- Also nearly always - resolving merge conflicts (https://help.github.com/en/articles/resolving-a-merge-conflict-using-the-command-line) (https://help.github.com/en/articles/resolving-a-merge-conflict-on-github)


In case of fire
1. git commit
2. git push
3. leave building

Thank you for being part of this class!

Roman Plessl

roman.plessl@prunux.ch

https://github.com/rplessl