

# Dive in to Git and GitHub

## Zoom Edition

<https://compicampus-git-intro.website>



CompiCampus

2021-10-04

Roman Plessl



# Agenda and Goals for Today

1. Git - Introduction to Version Control Systems , Git  and GitHub 
2. Learning Path and Exercises based on your knowledge and experience with Git and GitHub
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. How to use GitHub (II):
  - a. Host your personal, organization, and project sites with GitHub Pages.
  - b. Issues Tracking and Feature Requests, Actions, Task Management and Wiki, and more :)

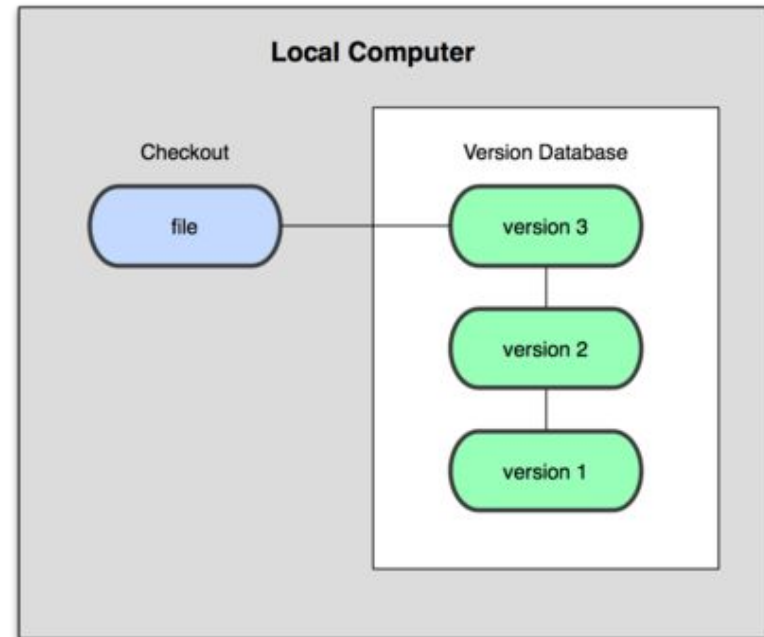
---

# Introduction to Distributed Version Control Systems and Git

# What is a version control system?

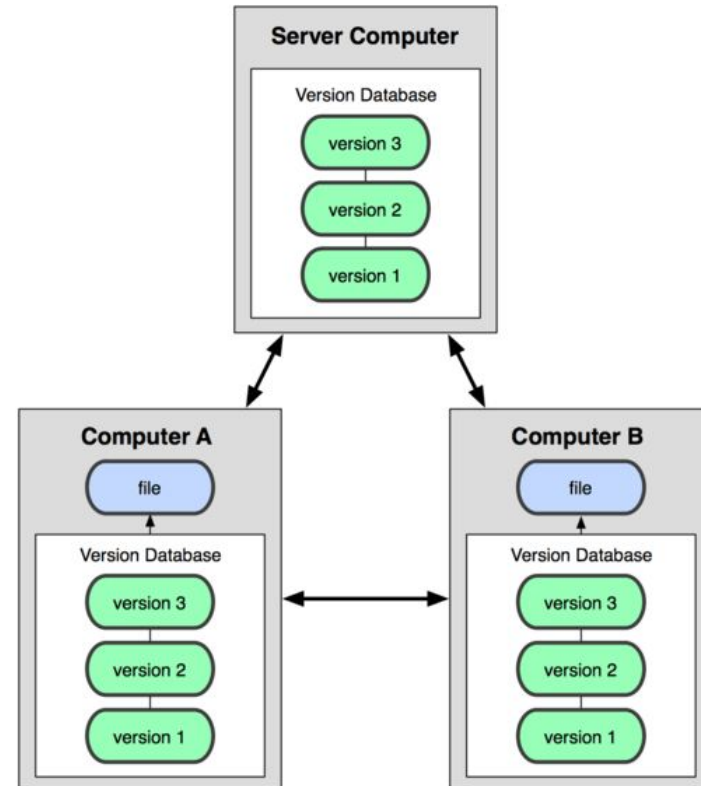
Version control system (VCS) in general

- 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)



# 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)

## Daten und Zahlen zu den jüngsten Versionen des Linux-Kernels

Kernel-Version	Anzahl Dateien <sup>1</sup>	Zeilen Quelltext (Ohne Doku) <sup>2</sup>	Entwicklungs-zeitraum	Commits (Ohne Merges) <sup>3</sup>	Diffstat <sup>4</sup>
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, 1.189.832 insertions(+), 600.665 deletions(-)

<sup>1</sup> `git ls-tree -r --name-only HEAD | wc -l`

<sup>2</sup> `find . -type f -not -regex "\.\/.git\/.*" | xargs cat | wc -l; echo "$ (find . -name *.[hcS] -not -regex "\.\/.git\/.*" | xargs cat | wc -l) "`

<sup>3</sup> `git-log --pretty=oneline vx.(y-1)..vx.(y) | wc -l; echo "($(git-log --pretty=oneline --no-merges vx.(y-1)..vx.(y) | wc -l))"`

<sup>4</sup> `git diff --shortstat vx.(y-1)..vx.(y)`

## Facts and Figures on the amount of Linux Kernel Changes

(creator [Thorsten Leemhuis](#), [heise online](#))

# 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** ([featured Projects](#))
- **Git** is now on Version 2.33

# 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** has a cloud based solution: github.com, Enterprise Version can be hosted on-prem.
- **GitHub** was build up in 2008 and bought 2018 by Microsoft (for 6.4 Mrd €)



# GitLab



- **GitLab** delivers a full DevOps life cycle from planning to creation, build, verify, security testing, deploying, and monitoring.
- **GitLab** allows to self-hosted the suite so code, project data and intellectual properties can be kept and secured in self-controlled perimeter: ETHZ, UZH, Switch, ... and many departments and institutions run their self-hosted GitLab Instance (e.g. <https://gitlab.ethz.ch>)
- **Gitlab** runs GitLab.com, a freemium product.
- **GitLab Inc.** was founded in 2014 by two Ukrainian developers and was founded multiple times by different venture capital companies.
- **GitLab Inc.** is a fully remote company, with over 1200 employees around the globe.

# 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** has a cloud based solution: github.com, Enterprise Version can be hosted on-prem.
- **GitHub** was build up in 2008 and bought 2018 by Microsoft (for 6.4 Mrd €)

# Snapshots and Commits

- The way git keeps track of your code history
- Essentially records what all your files look like at a given point in time
- You decide **when** to take a snapshot and of **what** files
- Have the ability to go back to visit any snapshot

## Commits:

- The act of creating a snapshot
- Essentially, a project is made up of a bunch of commits
- Commits contain three pieces of information:
  - a. Information about how the files changed from previously
  - b. Reference to the commit that come before it (called the parent node)
  - c. Hash code name (looks like: edfec504eb864dc557f3f5b9d3d301617036d15f3a)
- Commits as small as possible or as big as necessary

---

# Introduction to GitHub

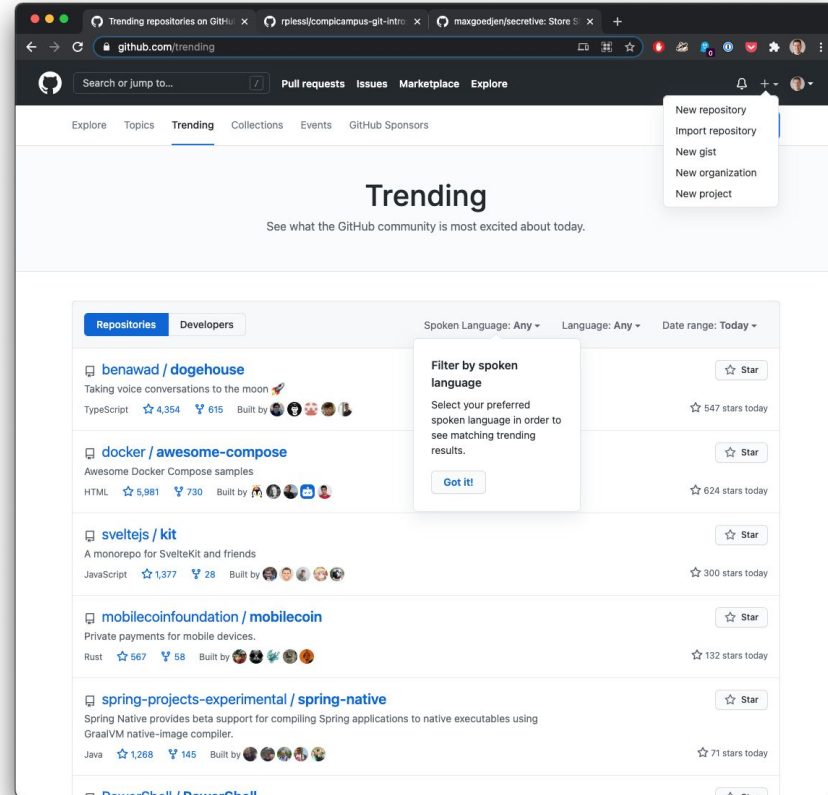
# Introduction to GitHub

<https://github.com>

Examples:

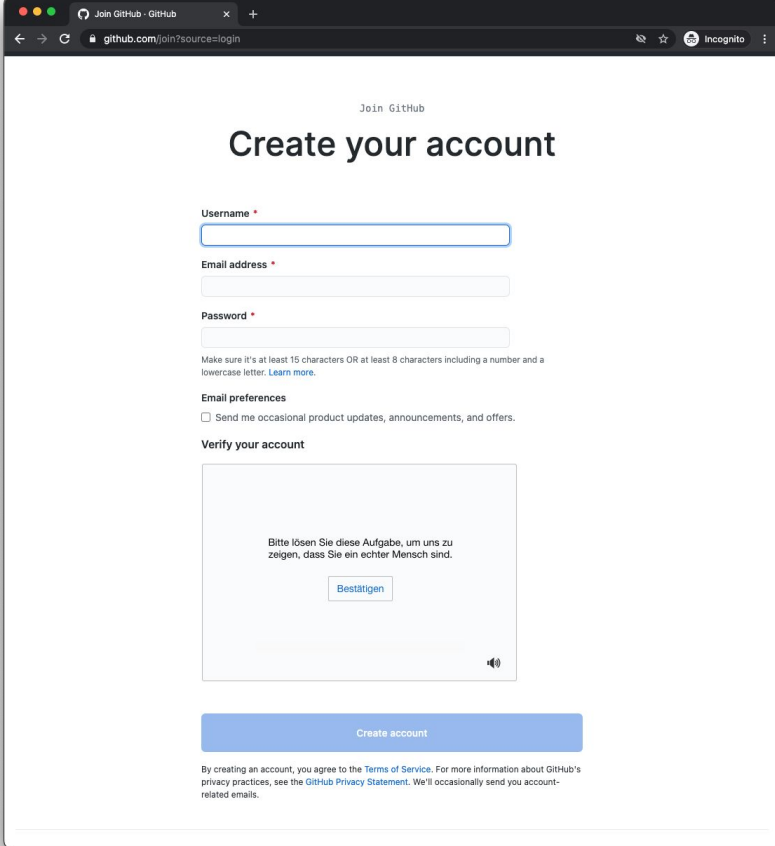
<https://lab.github.com/>

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



# Introduction to GitHub

<https://github.com>



The screenshot shows the GitHub 'Join GitHub' page in a web browser. The page title is 'Join GitHub' and the main heading is 'Create your account'. The form includes fields for 'Username', 'Email address', and 'Password'. Below the password field, there is a note: 'Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)'. Under 'Email preferences', there is a checkbox labeled 'Send me occasional product updates, announcements, and offers.' followed by the text 'Verify your account'. A large light blue box contains the German text: 'Bitte lösen Sie diese Aufgabe, um uns zu zeigen, dass Sie ein echter Mensch sind.' and a 'Bestätigen' button. At the bottom of the form is a large blue 'Create account' button. Below the button, a small disclaimer states: 'By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.'

---

# Learning Path and Exercises

# Virtual Classroom

- Git and GitHub Exercises
  - Information Slides are blue
  - Exercise Slides are red
- You could work in Local Breakout Sessions together or alone
  - You can (re-)join yourself to the a breakout room
  - I will create these 30 Breakout Rooms:
    - Git Beginners
    - GitHub Beginners
    - GitHub Intermediate Users
    - 20 additional rooms Room A - Room Z free to use
    - and additionally the main session stays alive
  - If there is a question worth to know for all:  
  
-> I will send a broadcast message and answer that question in the main session

# Based on your level with Git / GitHub:

My Idea of a Learning-Path:

1. See and touch how GitHub looks like and could be used
2. Touch and see how Git can be used locally - using git commands in command line:  
Basic Git Commands and Basic Git Workflow
3. GitHub
  - A. Setup your GitHub Environment
  - B. New Repositories can be done by hand and using GitHub in a Basic way
  - C. How to use Git together with GitHub (incl. Branching and Merging)
  - D. Collaborative Working and Coding GitHub

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

- Install Git ([p. 18](#)), Config ([p. 20](#)), Basic Git Workflow ([p. 23](#)) and Basic Git Commands ([p. 28](#))
- then with the How To Use GitHub ([p. 35](#)).

*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. 33](#)) and Git Architecture ([p. 34](#))
- GitHub Setup ([p. 35](#)) and local-remote Exercises ([p. 41 and ff](#))
- Git Branching, Merging and Rebase Exercise and Riddles ([p. 48](#))

**More Intermediate** Git and GitHub Users should start with:

- Understanding Git Cheat-Sheet ([p. 33](#)) and Git Architecture ([p. 34](#))
- GitHub and local-remote Git Exercises ([p. 41 and ff](#))
- Git Branching, Merging and Rebase Exercise and Riddles ([p. 48](#))
- Collaborative Working ([p. 50](#)) and GitHub Feature like Actions ([p. 58](#))
- Setup a Website with GitHub Pages ([p. 53](#))

---

# Install Git locally

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

mac OS (preinstalled):

- <https://git-scm.com/download/mac>
- I recommend to choose the Homebrew way - install [homebrew](#) and then `brew install git`

Windows (sometimes preinstalled, check if `git status` works when you open “`cmd.exe`”):

- Install <https://git-scm.com/download/win>
- or use WSL 2 <https://docs.microsoft.com/en-us/windows/wsl/install-win10>
- or use Windows Shell Integration <https://tortoisegit.org/>
- For SSH and SSH Agent Help follow: <https://cutt.ly/dhEnlH5> and <https://cutt.ly/KhEnQDL>

Linux (preinstalled):

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

---

# Basic Configuration of local Git

# Global Settings for locally installed Git

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). Now let's tell git things about you and your preferred settings:

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

```
# ~/.gitconfig  
[user]  
    name = Hans Muster  
    email = hans.muster@prunux.ch
```

See Customizing Git: <https://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration>

# Depending on your wished / needed Editor

## Here: Install Visual Studio Code and Change Editor for Git

Visual Studio Code has Git commands built-in.

- Usable on all platforms (mac, win, linux)
- Install Visual Studio Code from <https://code.visualstudio.com/>

Git needs an Editor to submit Changes, normally - if not set differently - this is `vi` !

By default, Git uses whatever you've set as your default text editor via one of the shell environment variables `VISUAL` or `EDITOR`, or else falls back to the `vi` editor to create and edit your commit and tag messages. To change that default to something else, you can use the `core.editor` setting:

```
$ git config --global core.editor "code --wait"
```

Now, no matter what is set as your default shell editor, Git will fire up Code to edit messages.  
(See also [Hints on Stack-Overflow](#))

---

# 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.

In GitHub you can add a new file -> type **“.gitignore”** in the text field and -> choose best fitting template.

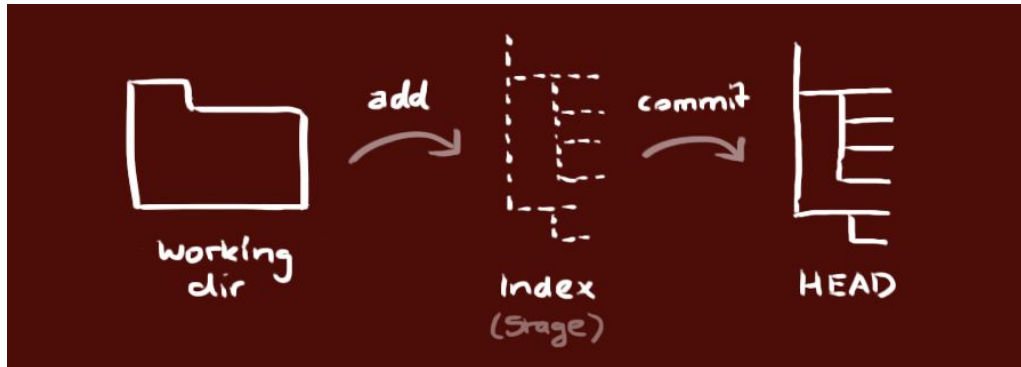
For further detail look at these examples and helping tool:

- <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

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>



# Git Cheat Sheet

From GitHub:

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

From GitLab:

<https://about.gitlab.com/images/press/git-cheat-sheet.pdf>

GitHub

GIT CHEAT SHEET

V1.11

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

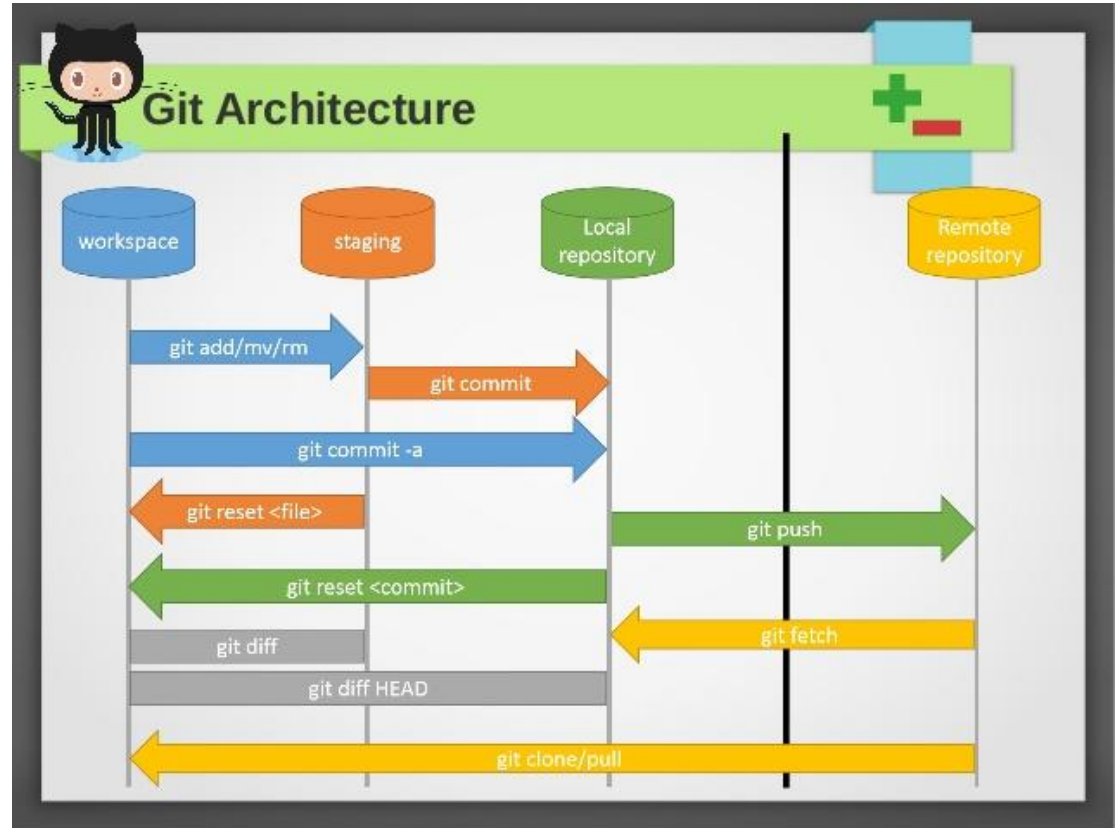
```
$ git merge [branch]
```

Combines the specified branch's history into the current branch

```
$ git branch -d [branch-name]
```

Deletes the specified branch

# Git Architecture



---

# 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)

The screenshot shows the GitHub homepage. At the top is a dark navigation bar with the GitHub logo, a search bar, and links for Pull requests, Issues, Marketplace, and Explore. A notification banner states "Your email was verified." Below this, the main content area features a large central card with a light blue and green gradient background. The card contains the text "Learn Git and GitHub without any code!" followed by instructions: "Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request." At the bottom of the card are two buttons: "Read the guide" (green) and "Start a project" (light blue). To the left of the card, under the heading "Repositories", it says "Your most active repositories will appear here." and provides links to "Create a repository" and "explore repositories". To the right, a "Discover repositories" section lists three projects: "frappe/erpnext" (Open Source ERP built for the web, Python, 4.3k stars), "box/box-ui-elements" (Box UI Elements, JavaScript, 78 stars), and "dotnet/docs.es-es" (PowerShell, 42 stars). A "Go to Explore" link is at the bottom right. A small welcome message box is also visible in the top right of the main content area.

Search or jump to... Pull requests Issues Marketplace Explore

Your email was verified.

### Repositories

Your most active repositories will appear here.  
[Create a repository](#) or [explore repositories](#).

## Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

[Read the guide](#) [Start a project](#)

Welcome to the new dashboard. Get closer to the stuff you care about most.

### Discover repositories

**frappe/erpnext**  
Open Source ERP built for the web  
Python ★ 4.3k

**box/box-ui-elements**  
Box UI Elements  
JavaScript ★ 78

**dotnet/docs.es-es**  
PowerShell ★ 42

[Go to Explore](#) →

# 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 “my-compicampus-hello-world” 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>



---

# 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

# 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**

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

```
# change into the `compicampus-<USERNAME>` directory  
cd compicampus-<USERNAME>
```

```
# 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 `README.md` and `index.html` using
# the text editor (like visual studio code, nano, vim,... )

# stage the changed files
git add README.md index.html

# 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
```

# 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>

---

# Git is operating on branches



# Working with Git branches

**Branching:** After the basic using Git and GitLab 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](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 preferred 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, Actions, Task  
Management and Wiki, and more**



# Collaborative Features included in GitHub

GitHub includes several collaborative features for collaborative coding, automation & CI/CD, Security, Project Management and so one.

- All Features in one big page

<https://github.com/features>

- 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

Automation and CI/CD:

- GitHub Actions: Automate all your software development workflows. Write tasks and combine them to build, test, and deploy faster from GitHub

<https://docs.github.com/en/actions>

- Package and Release Code

<https://docs.github.com/en/packages>

---

# Further Reading and Exercises

# Git / GitHub Specialites

## Git Large File Storage (LFS):

Large Git repositories become very slow and put a lot of strain on the GitHub server. The Large File Storage (LFS) extension is used to off-load large files from a repository to a separate storage.

<https://docs.github.com/en/github/managing-large-files>

## Protected Branches:

- <https://docs.github.com/en/github/administering-a-repository/about-protected-branches>

# 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>
- A good explanation of how to git reset a checkout to a previous version is described here:  
<https://www.atlassian.com/git/tutorials/undoing-changes/git-reset>

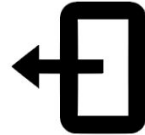
## In case of fire



1. `git commit`



2. `git push`



3. leave building

# Thank you for being part of this class!

Roman Plessl

[roman.plessl@prunux.ch](mailto:roman.plessl@prunux.ch)

<https://github.com/rplessl>

