Dive in to Git and GitLab

Physical Edition

https://compicampus-git-intro.website

CompiCampus

2022-10-17

Roman Plessl

Plessl + Burkhardt

nuvibit

Agenda and Goals for Today

1.   Git - Introduction to Version Control Systems and Git

2.   Introduction to GitLab (I): Overview, Documentation - Create or Login to your GitLab Account, Create a new Project, Using a Repository, General Settings (public vs. private)

3.   Learning Path and Exercises based on your knowledge and experience with Git / GitLab including a bunch of practical exercises:
     a.   Basic Git Workflow, Basic Git Commands
     b.   Using GitLab (II), including: Login or Create a GitLab Account, Create a new Project, Using Repositories and How to use local Git with GitLab
     c.   Working with Git branches
     d.   Interact with each other's and other coders: Git Branching, Code Changes and Commits, Pull and Merge Requests, Comments

4.   How to use GitLab (III):
      Intro to CI/CD {gitlab-ci}, Issue Tracking, Wiki, GitLab Pages, Settings

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

- 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 (stable) states**
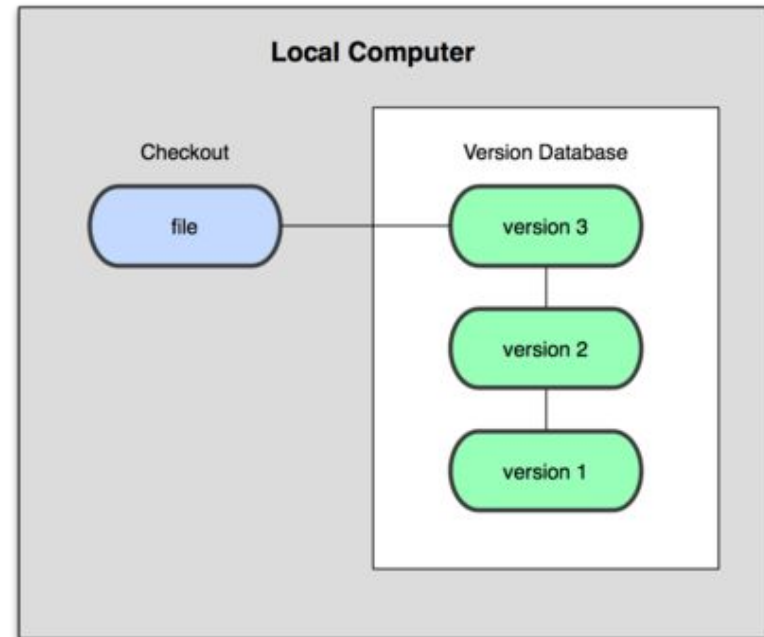
What is <u>distributed</u> version control?

- 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 access to the internet was quite different >17 years ago)
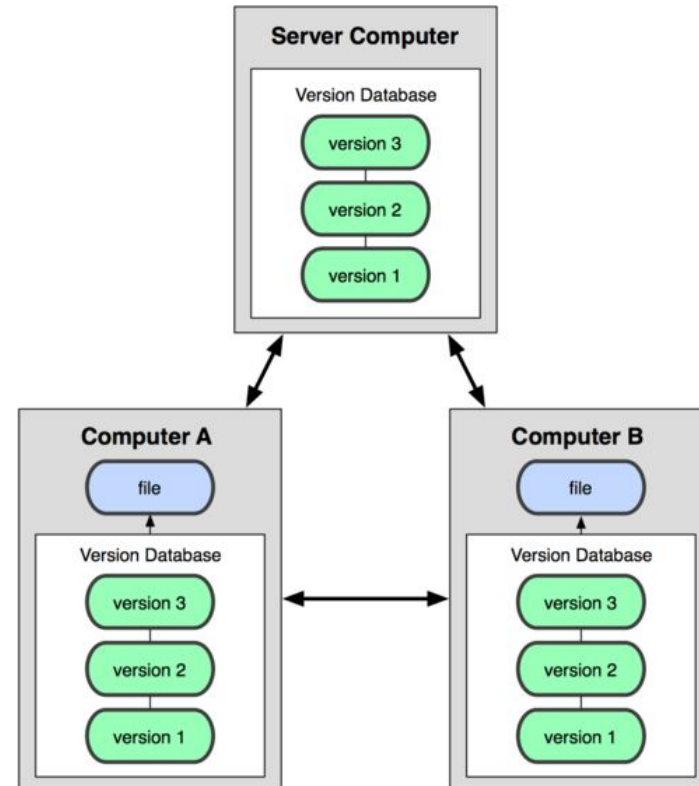
History about Git

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)

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

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 "($(find . -name *.[hcS] -not -regex '\./\.git/.*' | xargs cat | wc -l))"
[3] git-log --pretty=oneline vx.(y-1)..vx.(y) | wc -l; echo "($(git-log --pretty=oneline --no-merges vx.(y-1)..vx.(y) | wc -l))"
[4] git diff --shortstat vx.(y-1)..vx.(y)

Kernel Stats (by Colin Ian King, document licensed by CC-BY 4.0)
https://docs.google.com/spreadsheets/d/1_yH7lFmZxAoSWrtsd8tGu3befG4zIcMnytB1ml4pQQM/edit#gid=0

Git

GitLab

- **Git** is a (*the*) 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 v/o Jun Hamano.

- Over 90% of all Code Projects use **Git** (featured Projects)

- **Git** is now on Version 2.38

- **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, since last year with Public Stocks.

- **GitLab Inc.** is a fully remote company, with over 1400 employes around the globe.

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, since last year with Public Stocks.

- **GitLab Inc.** is a fully remote company, with over 1400 employes 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, issues / feature / bug tracking, 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 -> Commits

- Snapshots is 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 are / should be as small as possible or as big as necessary

Introduction to GitLab

Introduction to GitLab
https://gitlab.com
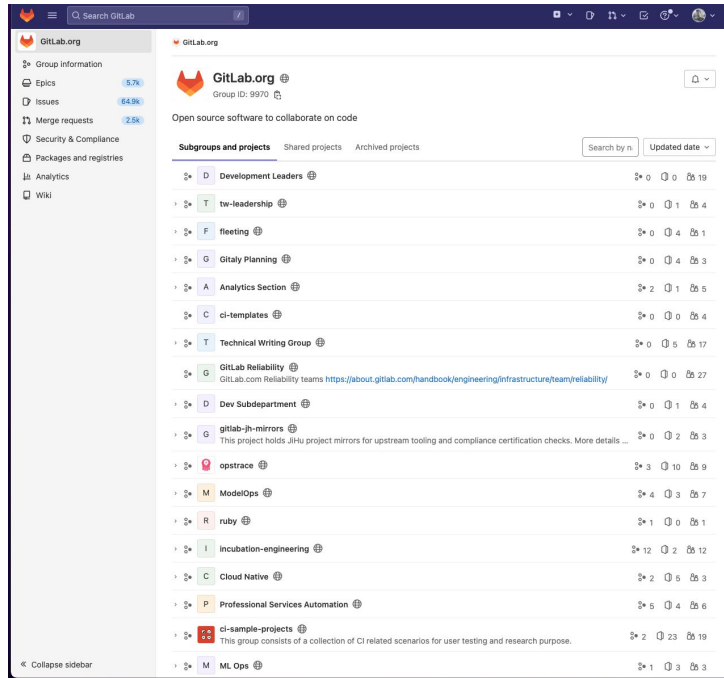
SWITCH edu-ID

Help   EN ∨

Log in to: gitlab.uzh.ch

Service description:
GitLab is an open source end-to-end software development platform with built-in version control, issue tracking, code review, CI/CD, and more.

SWITCH edu-ID

E-mail: john.doe@example.org
Password: Enter your password   👁

Create account   Login

Forgot password?
Options for personal data protection

About / Terms of Use / Legal Notice / Imprint

SWITCH

The ... GitLab instance is behind the Switch edu-ID
Single-Sign-On, so all users must have an edu-ID currently.

Introduction to GitLab

https://gitlab.com

https://gitlab.com/explore/projects/trending

Example GitLab Projects:

https://gitlab.com/gitlab-org/gitlab-foss

https://gitlab.com/wireshark/wireshark

https://gitlab.com/tortoisegit/tortoisegit

Introduction to GitLab
https://gitlab.com

GitLab.com

GitLab.com offers free unlimited (private) repositories and unlimited collaborators.

- **Explore projects on GitLab.com** (no login needed)
- **More information about GitLab.com**
- **GitLab Community Forum**
- **GitLab Homepage**

By signing up for and by signing in to this service you accept our:

- **Privacy policy**
- **GitLab.com Terms**.

Username or email

Password

☐ Remember me   Forgot your password?

Sign in

Don't have an account yet? Register now

Sign in with

- G Google
- GitHub
- Twitter
- Bitbucket
- Salesforce

☐ Remember me

Introduction to GitLab

Quote from the former self paced
learning plattform of GitHub

Learning should be fun
There are no simulations or boring tutorials here, just
hands-on lessons created with ♥ by the GitHub
community and taught by the friendly Learning Lab bot.

Real projects
Learn new skills while working in your
own copy of a real project.

Helpful bot
Our friendly bot provides instructions
and feedback throughout your journey.

Real workflow
Everything happens in GitHub Issues
and Pull Requests.

Learning Path and Exercises

Exercises & Support

- Git and GitLab Exercises
    - ■ <u>Information</u> Slides are in blue
    - ■ <u>Exercise</u> Slides are in red

- You could work in alone or together in small groups with the following levels:
    - ■ Git Beginners
    - ■ GitLab Beginners
    - ■ GitLab Intermediate Users

- Raise your hand if you have questions or need help

Based on your level with Git / GitLab:

My Idea of a Learning-Path:

1. See and touch how GitLab 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. GitLab   A. Setup your GitLab Environment

              B. Sandboxes and Hello World Repository Examples

              C. How to use Git together with GitLab (incl. Branching and Merging)

              D. Collaborative Working with GitLab & CI/CD with GitLab

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

- Install Git (p. 19), Config (p. 21), Basic Git Workflow (p. 24) and Basic Git Commands (p. 29)
- then with the How To Use GitLab (p. 37).

Intermediate and Advanced users, see next slide.

Based on your level with Git / GitLab:

Intermediate Git and GitLab Users should start with:

- Understanding Git Cheat-Sheet (p. 34) and Git Architecture (p. 35)
- Using GitLab and local-remote Exercises (p. 37) and (p. 45)
- Git Branching, Merging and Rebase Exercise and Riddles (p. 52)

More Intermediate Git and GitLab Users should start with:

- Understanding Git Cheat-Sheet (p. 34), Git Architecture (p. 35) and Git Reset Mechanism (p. 36)
- Git Branching, Merging and Rebase Exercise and Riddles (p. 52)
- Using GitLab and local-remote Exercises (p. 37) and (p. 45)
- Collaborative Interaction (p. 54)
- GitLab CI/CD (p. 57)
- GitLab Pages (p. 63), Issue Tracking (p. 66) , and Wiki (p. 67)

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 the WSL 2 https://docs.microsoft.com/en-us/windows/wsl/install-win10
- or use the 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 a `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 GitLab)

### onmac, linux, wsl2 or git-bash.exe)
# cat ~/.gitconfig (mac, linux, wsl2 or git-bash.exe)
# [user]
#    name  = Hans Muster
#    email = hans.muster@prunux.ch
```

(Further Info 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

My preferred editor is Visual Studio Code which 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 should use the core.editor setting:

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


# or using your known terminal editor like
# git config --global core.editor "nano" OR git config --global core.editor "vim"
```

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 for improve Visual Studio Code integration if not working

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 GitLab 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.
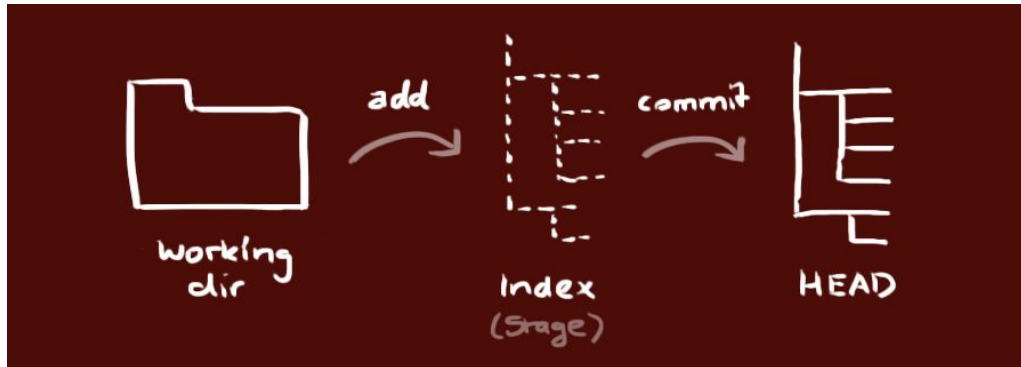
Basic Git Commands

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 GitLab:

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

From GitHub:

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

Git Cheat Sheet

01 Git configuration

`$ git config --global user.name "Your Name"`
Set the name that will be attached to your commits and tags.

`$ git config --global user.email "you@example.com"`
Set the e-mail address that will be attached to your commits and tags.

`$ git config --global color.ui auto`
Enable some colorization of Git output.

02 Starting A Project

`$ git init [project name]`
Create a new local repository. If [project name] is provided, Git will create a new directory name [project name] and will initialize a repository inside it. If [project name] is not provided, then a new repository is initialized in the current directory.

`$ git clone [project url]`
Downloads a project with the entire history from the remote repository.

03 Day-To-Day Work

`$ git status`
Displays the status of your working directory. Options include new, staged, and modified files. It will retrieve branch name, current commit identifier, and changes pending commit.

`$ git add [file]`
Add a file to the **staging** area. Use in place of the full file path to add all changed files from the **current directory** down into the **directory tree**.

`$ git diff [file]`
Show changes between **working directory** and **staging area**.

`$ git diff --staged [file]`
Shows any changes between the **staging area** and the **repository**.

`$ git checkout -- [file]`
Discard changes in **working directory**. This operation is **unrecoverable**.

`$ git reset [file]`
Revert your **repository** to a previous known working state.

`$ git commit`
Create a new **commit** from changes added to the **staging area**. The **commit** must have a message!

GitLab | everyone can contribute · · · about.gitlab.com

`$ git rm [file]`
Remove file from **working directory** and **staging area**.

`$ git stash`
Put current changes in your **working directory** into **stash** for later use.

`$ git stash pop`
Apply stored **stash** content into **working directory**, and clear **stash**.

`$ git stash drop`
Delete a specific **stash** from all your previous **stashes**.

04 Git branching model

`$ git branch [-a]`
List all local branches in repository. With -a: show all branches (with remote).

`$ git branch [branch_name]`
Create new branch, referencing the current **HEAD**.

`$ git checkout [-b][branch_name]`

05 Review your work

`$ git log [-n count]`
List commit history of current branch. -n count limits list to last n commits.

`$ git log --oneline --graph --decorate`
An overview with reference labels and history graph. One commit per line.

`$ git log ref..`
List commits that are present on the current branch and not merged into **ref**. A **ref** can be a branch name or a tag name.

`$ git log ..ref`
List commit that are present on **ref** and not merged into current branch.

`$ git reflog`
List operations (e.g. checkouts or commits) made on local repository.

Git Architecture
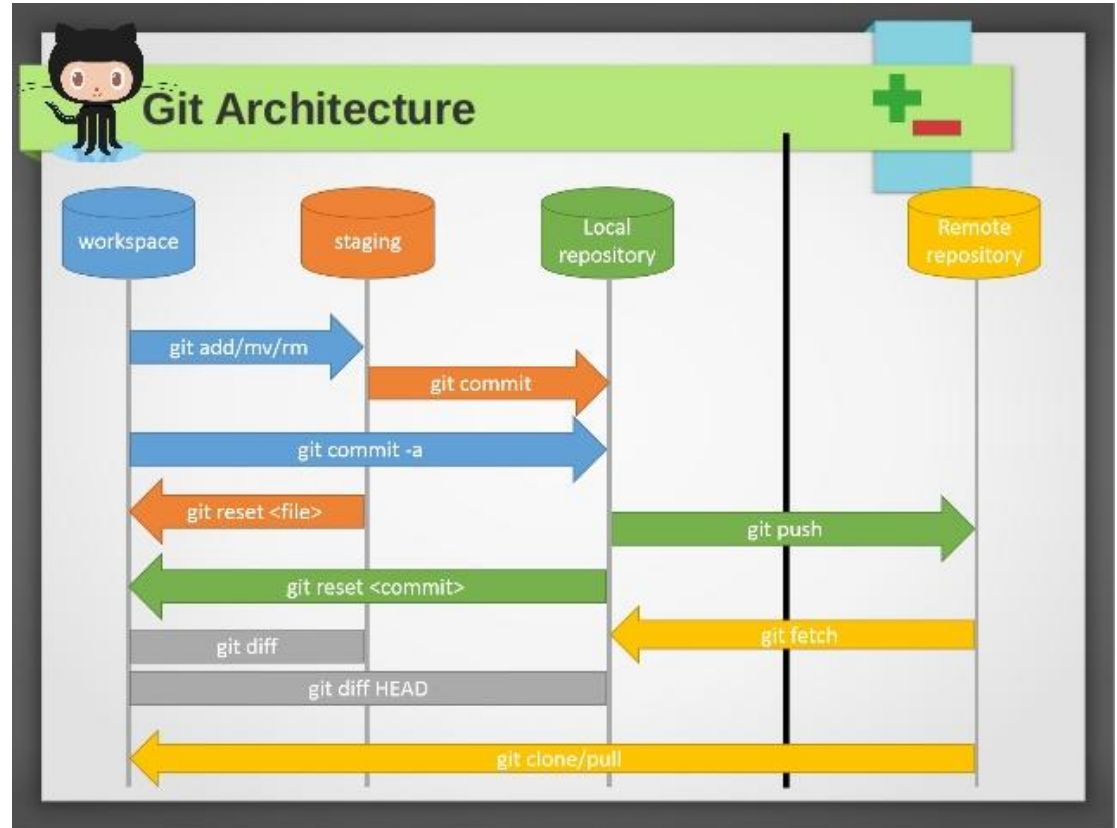
Advanced: How to "git reset" (spare time)

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 GitLab (I)

Useful Links

Gitlab user documentation: http://doc.gitlab.com/ce/

Gitlab flavoured markdown documentation: https://docs.gitlab.com/ce/user/markdown.html

Gitlab Groups: https://docs.gitlab.com/ee/user/group/

Gitlab CI/CD: https://docs.gitlab.com/ee/ci/README.html

Git link-list from GitLab: https://docs.gitlab.com/ee/topics/git/

How to use GitLab (I): Create a GitLab Account, create new projects (Variant I)

If you haven't already a GitLab account -> create one:

- Choose a not already taken username for the account and fill in your personal data in "Register now" or Sign in with an Identity Provider (Google, Github, Twitter, …) .

- To complete, solve the recaptcha riddle.

- Choose your role

- Verify your Email Address with your Email Account. Email from gitlab.com to verify your identity.

- GitLab Free Account is sufficient for most cases.

- Create a new Project (green button) -> "gitlab-intro-sandbox"

How to use GitLab (I): Create a GitLab Account, create new projects (Variant II)

If you haven't already logged in to your UNIVERSITY GitLab account:

- Login to https://gitlab.MYUNIVERSITY.ch

- Login with your EDU-ID Account.

- Create a new Project (green button) -> "gitlab-intro-sandbox"

Set a User Account Password:

⚠ You won't be able to pull or push project code via SSH until you add an SSH key to your profile

[Add SSH key] [Don't show again]

⚠ You won't be able to pull or push project code via HTTPS until you set a password on your account

[Remind later] [Don't show again]

How to use GitLab (I): <u>Optional</u>
Create a free gitlab.com account to test CI/CD pipelines

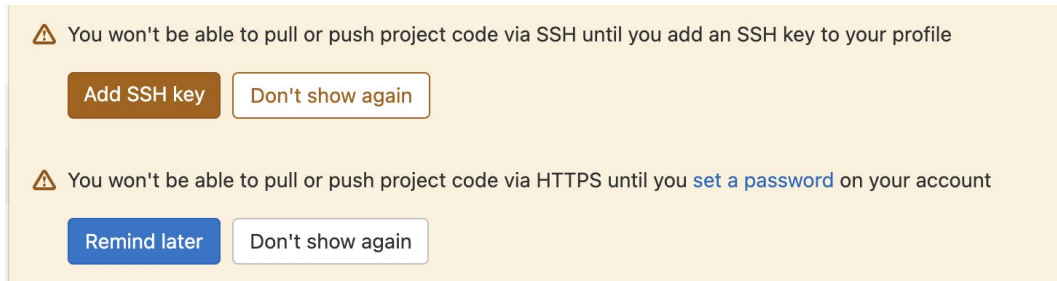If you like to experiment more (e.g. with CI/CD) use or create a free gitlab.com account:

- Choose a not already taken username for the account and fill in your personal data in "Register now" or Sign in with an Identity Provider (Google, Github, Twitter, ...) .

- To complete, solve the recaptcha riddle.

- Choose your role

- Verify your Email Address with your Email Account. Email from gitlab.com to verify your identity.

- GitLab Free Account is sufficient for most cases.

Create your first sandbox example's (I)
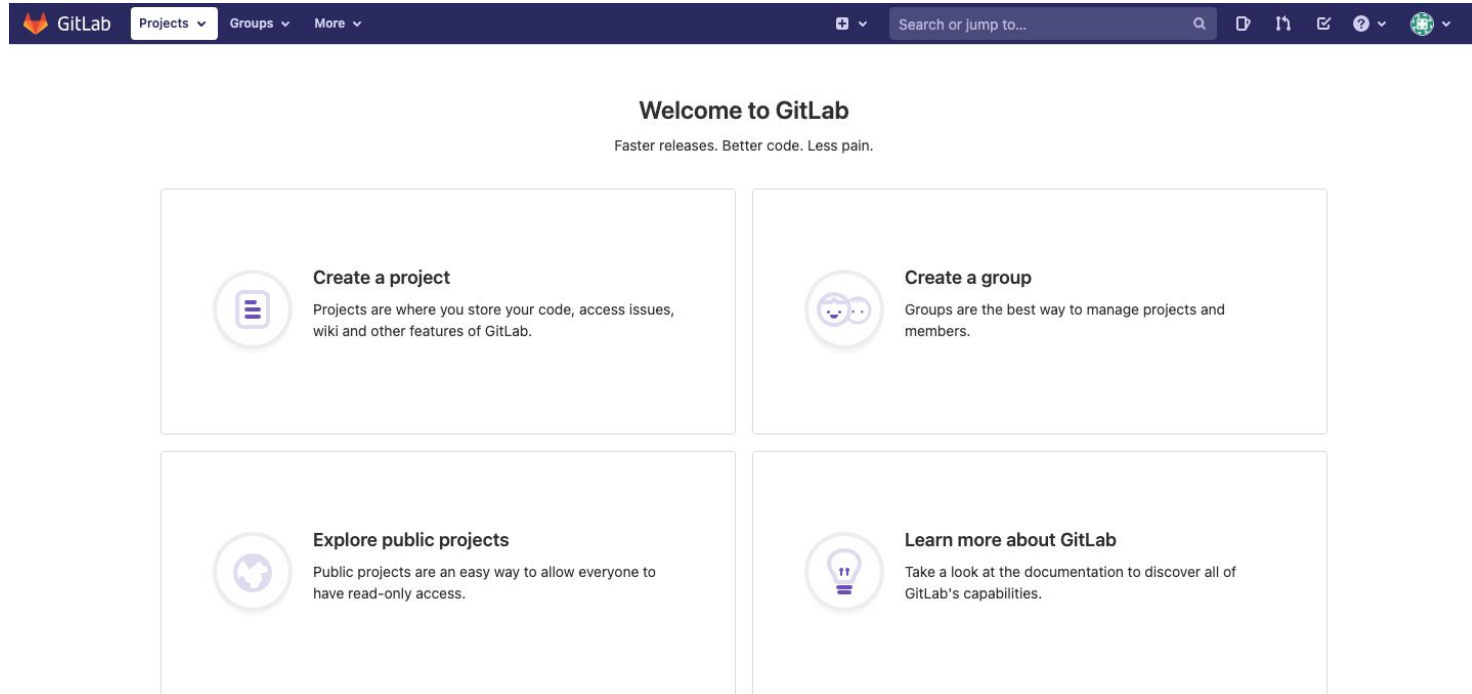
Create your first group, sub-group and sandbox environment filled with data (II)

Create New Group and Subgroup

- Create a "New group" with the "+" sign on top with the name

  "`git-gitlab-course-USERNAME`" (visibility internal)

- Create a "New subgroup" with the name "`sandboxes`" (visibility internal)

Add Sample Project:

- Create a new Project in this subgroup -> "New project"

- Choose the Tab: "Create from template" -> and Buit-In "Sample GitLab Project"

- Choose "Basic" by clicking on "use template"; name your project "`sandbox-basic`" (visibility private)

Create your first empty project (III):
`hello-world`

We will create a "hello-world" Project and Content

- Create a new Project -> "New project"

- Choose the Tab: "Blank Project"

- Name the new Project "`hello-world`"

- Check in the Project URL, that you have chosen the correct Subgroup "**git-gitlab-course-abc/sandbox**"

- Visibility should be private (we will add additional workmates later)

- Check "`Initialize repository with a README`"

Modify your first empty project (IV):
`hello-world`

- create a new branch (look at the picture on the website)

  https://docs.gitlab.com/ee/user/project/repository/web_editor.html#create-a-new-branch

- edit your README.md file and enhanced the content with Markdown Text (Click on the REAMDE.md -> Click on "Edit" in the Context-Menu")

  https://docs.gitlab.com/ce/user/markdown.html
  https://www.webfx.com/tools/emoji-cheat-sheet/

- create a merge request (look at the picture on the website)

  https://docs.gitlab.com/ee/user/project/merge_requests/creating_merge_requests.html

- merge the branch to the master branch

How to use local Git with GitLab

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

- Create a new Project on GitLab "`hello-world-locally`" (using the Project URL with the "sandbox" subgroup, Visibility should be private, without Initialize README)

- Re-open the shell or prompt with the git command line interface

- Clone your new GitLab Repository locally with **HTTPS** as documented here
  ```
  git clone https://gitlab.com/USERNAME/git-gitlab-course-USERNAME/sandboxes/hello-world-locally.git
  cd hello-world-locally
  touch README.md
  git add README.md
  git commit -m "add README"
  git push -u origin master
  ```

- Delete the Project on GitLab as documented here:
  https://docs.gitlab.com/ee/user/project/settings/#delete-a-project

- Delete your local checkout

How to use local Git with GitLab (II): Cloning using HTTPS
Exercise: Contribute to an existing repository

- **Contribute to an existing repository**

  ```
  # clone a repository on GitLab.com to your machine
  git clone https://gitlab.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

```
# 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 "add the WHY I MADE THIS CHANGES"

# for security (MFA) you need a personal access token to push, so create one now:
# https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html#create-a-personal-access-token
# with the 'write_repository' grant right
# https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html#personal-access-token-scopes

# push changes to gitlab
git push --set-upstream origin my-branch

# afterwards that code changes can be merged by a "merge request"
```

How to use local Git with GitLab (IV): Push Content using HTTPS
Exercise: Start a new repository and publish it to GitLab

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

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

    # Now create add code / files with the command line

    ```
    # create a new directory, and initialize it with git-specific functions
    git init --initial-branch=main 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 GitLab (V): Push Content using HTTPS
Exercise: Start a new repository and publish it to GitLab

```
# 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 GitLab
git remote add origin https://gitlab.com/git-gitlab-course-USERNAME/...

# push changes to gitlab
git push --set-upstream origin main
```

Git is working with 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

A very popular Version of using Git Branches (by meaning full names `main aka master, develop, hotfix, feature`) is documented here by the inventor:

- https://nvie.com/posts/a-successful-git-branching-model/

Interact with each other and other coders

Interact with each others and other coders

Forking: After using Git and GitLab 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://docs.gitlab.com/ee/user/project/repository/forking_workflow.html#creating-a-fork

- Fork this repository to your own space:
  `https://gitlab.com/git-gitlab-course-main/example-repo`

- Clone the forked repository to your local machine:
  `git clone https://gitlab.com/` … `(copy the URL)`

- 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 GitLab

- Create a merge request

Interact with each others and other coders

Merge Requests, Comments on Merge 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 GitLab Account username and for a Project name
3. Fork the Project of your colleague on GitLab
4. Clone it to your local machine
5. Change some code and text parts in the Project
6. Commit your changes
7. Push it to GitLab back
8. Create a merge request
   https://docs.gitlab.com/ee/user/project/merge_requests/
9. And comment parts of the code changes in the merge request to you
   https://docs.gitlab.com/ee/user/discussions/index.html#merge-request-reviews

GitLab CI/CD
Build, Integrate, Deliver and Deploy anytime

GitLab CI/CD

Usually, if you choose GitLab as your Git Frontend, you also choose GitLab especially to use the CI/CD pipeline features! GitLab CI/CD is a tool built into GitLab for software development through the continuous methodologies:

- Continuous Integration (CI)
- Continuous Delivery (CD)
- Continuous Deployment (CD)

Documentation: https://docs.gitlab.com/ee/ci/README.html

Example:

- https://gitlab.com/rplessl/compicampus-git-gitlab-intro

GitLab CI/CD

GitLab CI/CD is configured by a file called .gitlab-ci.yml placed at the repository's root. This file creates a pipeline, which runs for changes to the code in the repository. Pipelines consist of one or more stages that run in order and can each contain one or more jobs that run in parallel. These jobs (or scripts) get executed by the GitLab Runner agent.

To get started with GitLab CI/CD, we recommend you read through the following documents:

- Get started with GitLab CI/CD.

- Fundamental pipeline architectures.

- GitLab CI/CD basic workflow.

- Step-by-step guide for writing .gitlab-ci.yml for the first time.

GitLab CI/CD
Good Examples for CI/CD in University Context

- Automagically build LaTeX PDF by pushing source and using built PDF from Pipeline
    - https://blog.martisak.se/2020/05/11/gitlab-ci-latex-pipeline/
    - https://gitlab.com/git-gitlab-course-main/sandboxes/latex-pipeline
      (don't miss to add the Badge Section)

- Automatically build MATLAB function / Tests
    - https://stackoverflow.com/questions/34647154/gitlab-ci-with-matlab
    - https://github.com/mathworks-ref-arch/matlab-dockerfile

-

Specialities -> Git / GitLab

Git / GitLab Specialities

Git Large File Storage (LFS):

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

- https://docs.gitlab.com/ee/topics/git/lfs/index.html

Protected Branches:

- https://docs.gitlab.com/ee/user/project/protected_branches.html

Share Projects with Groups:

- https://docs.gitlab.com/ee/user/project/members/share_project_with_groups.html

GitLab Pages
Host your static website

Static Website with GitLab Pages

GitLab Pages are public webpages hosted and easily published through GitLab

- GitLab Pages are free of charge static websites
- The GitLab Page Website is build with one of these Static Site Generators: Mkdocs, Hugo, Jekyll, Gatsby, Middleman, Harp, Hexo, Brunch, ...
- Websites are themeable
- Content and Style can be modified remotely via the GitLab Interface or locally in the checkout on your computer
- Website Content is mostly given in Markdown
- A custom Website Address (Domain) can be chosen (incl. SSL Certificate from Letsencrypt)

Documentation: https://docs.gitlab.com/ee/user/project/pages/

Example: https://gitlab.com/rplessl/compicampus-git-gitlab-intro

Create your personal GitLab Pages Website

- Create your own GitLab Pages Site
  (at the moment not working on the nccr gitlab, so use your gitlab.com account for this exercise)

- Fork and enhance:
  https://gitlab.com/rplessl/compicampus-git-gitlab-intro(based on the easy usable mkdocs)

  Or build a new website with Static Website Generator

  Hugo is the next level of rather easy to use tools

Issues Tracking and Feature Requests, Wiki and Settings

Collaborative Projects Features

GitLab includes [several collaborative features](#) for handling, enhancing and reporting of the code / of the **project**.

- Issues and Kanban Board, Service Desk, Milestones

  https://docs.gitlab.com/ee/user/project/issues/index.html

  From my Point of View: The Issues List and Board is really intuitive and perfect integrated.
  The Service Desk is really cool to offer customer support through Email send and replied directly inside the GitLab Instance.

- Wiki

  Automatically Integrated Documentation to each project

Create Project Wiki and Issue Tracking

- Create your own GitLab Project Wiki
  (at the moment not working on the nccr gitlab, so use your gitlab.com account for this exercise)

- Create Issues in the Issue Tracking and use the Service Desk

Settings

- Project Settings:

  https://docs.gitlab.com/ee/user/project/settings/#project-settings

- User Profile Settings:

  https://docs.gitlab.com/ee/user/profile/#profile-settings

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

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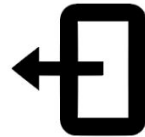 https://www.atlassian.com/git/tutorials/undoing-changes/git-reset

Thank you for being part of this class!

Roman Plessl

roman@plessl-burkhardt.ch

https://gitlab.com/rplessl