gitops-Training

Agenda

1. Installation (GIT)
    - GIT auf Ubuntu/Debian installieren
    - GIT unter Windows installieren
2. Kubernetes
    - Installation micro8ks (Ubuntu)
3. Commands (with tipps & tricks)
    - git add + Tipps & Tricks
    - git commit
    - git log
    - git config
    - git show
    - Needed commands for starters
    - git branch
    - git checkout - used for branches and files
    - git merge
    - git tag
4. Advanced Commands
    - git reflog
    - git reset - Back in Time
5. Docker
    - Install docker on Ubuntu
    - Important commands
6. github pages
    - Github Pages
7. github actions
    - General overview
    - Add a self-host runner
    - Create dependant jobs
    - Create custom composite action
    - Create custom docker action
    - Work with artefacts
    - Create digitalocean-kubernetes.md
    - Deploy to server with ssh
8. github actions - examples
    - Simple Workflow Test
    - Checkout Repo
    - Push to repo
9. Nix kaputtmachen - so gehts

- Die 5 goldenenen Regeln

10. Tips & tricks

    - Beautified log
    - Change already committed files and message
    - Best practice - Delete origin,tracking and local branch after pull request/merge request
    - Change language to german - Linux
    - Reference tree without sha-1
    - Always do pull --rebase for master branch

11. Exercises

    - merge feature/4712 - conflict
    - merge request with bitbucket

12. Snippets

    - publish lokal repo to server - bitbucket
    - failure-on-push-fix
    - failure-on-push-with-conflict

13. Extras

    - Best practices
    - Using a mergetool to solve conflicts

14. Help

    - Help from commandline

15. Documentation

    - GIT Pdf
    - GIT Book EN
    - GIT Book DE
    - Third Party Tools

Installation (GIT)

GIT auf Ubuntu/Debian installieren

Installation

```
sudo apt update
sudo apt install git
```

Language to english please !!

```
sudo update-locale LANG=en_US.UTF-8
su - kurs

## back to german

sudo update-locale LANG=de_DE.UTF-8
su - kurs

## Reference:
https://www.thomas-krenn.com/de/wiki/Locales_unter_Ubuntu_konfigurieren

## update-locale does a change in
$ cat /etc/default/locale
LANG=en_US.UTF-8
```

GIT unter Windows installieren

- https://git-scm.com/download/win

Kubernetes

Installation micro8ks (Ubuntu)

Reference:

- https://ubuntu.com/tutorials/install-a-local-kubernetes-with-microk8s#2-deploying-microk8s

Commands (with tipps & tricks)

git add + Tipps & Tricks

Trick with -A

```
## only adds from the folder you are in recursively
## but not above (you might miss some files, when you are in a subfolder
git add .

### Fix -A
## adds everything no matter in which folder you are in your project
git add -A
```

git commit

commit with multiple lines on commandline (without editor)

```
 git commit -am "New entry in todo.txt

* nonsene commit-message becasue of missing text-expertise"
## enter on last line
```

Change last commit-mesage (description)

```
git commit --amend
## now you can change the description, but you will get a new commit-id
```

git log

Show last x entries

```
##
## git log -x
## Example: show last 2 entries
git log -2
```

Show all branches

```
git log --all
## oder wenn alias alias.lg besteht:
## git lg --all
```

Show first log entry

```
## Step 1 - log needs to only show one line per commit
git log --oneline --reverse

## Step 2: combine with head
git log --oneline --reverse | head -1
```

Multiple commands with an alias

```
git config --global alias.sl '!git log --oneline -2 && git status'
```

git config

How to delete an entry from config

```
## Important: Find exact level, where it was added --global, --system, --local
## test before
## should contain this entry
git config --global --list
```

```
git config --unset --global alias.log
```

git show

Show information about an object e.g. commit

```
git show <commit-ish>
## example with commit-id
git show 342a
```

Needed commands for starters

```
git add -A
git status
git log  // git log -4 // or beautified version if setup as alias git lg
git commit -am  "commit message" // "commit message" can be freely chosen
## for more merge conflict resultion use only
git commit # to not change commit - message: must be message with merge
## the first time
git push -u origin master
## after that
git push
git pull
```

git branch

Create branch based on commit (also past commit)

```
git branch lookaround 5f10ca
```

Delete unmerged branch

```
git branch -d branchname # does not work in this case
git branch -D branchname # <- is the solution
```

git checkout - used for branches and files

Checkout (change to) existing branch

```
git checkout feature/4711
```

Checkout and create branch

```
## Only possible once
git checkout -b feature/4712
```

Recover deleted file

```
rm todo.txt
## get from last from last commit
```

```
git checkout HEAD -- todo.txt
```

git merge

Merge without conflict with fast-forward

```
## Disadvantage: No proper history, because only one branch visible in log
## after fast-forward - merge


## Important that no changes are in master right before merging
git checkout master
git merge feature/4711
```

Merge (3-way) also on none-conflict (no conflicts present)

```
git merge --no-ff feature/4711
```

git tag

Creating and using tags

```
## set tag on current commit -> HEAD of branch
git tag -a v1.0 -m "my message for tag"
## publish
git push --tags

## set on specific commit
git tag -a v0.1 -m "Initial Release" a23c

## checkout files of a specific tag
git checkout v0.1
## or
git checkout tags/v0.1
```

Deleting tags

```
## Fetch new tags from online
git fetch --tags

## Update master branch (rebase) and fetch all tags in addition from online
git checkout master
git pull --rebase --tags

## Tag local löschen und danach online löschen
git tag -d test.tag
git push --delete origin test.tag

## Tag online löschen und danach lokal
## Schritt 1: Über das interface (web) löschen
## Schritt 2: aktualisieren
```

```
git fetch --prune --prune-tags
```

Advanced Commands

git reflog

command

- show everything you (last 30 days), also stuff that is not visible in branch anymore

Example

```
git reflog
```

when many entries a pager like less (aka man less) will be used

```
## you can get out of the page with pressing the key 'q'
```

git reset - Back in Time

Why ?

- Back in time -> reset
- e.g. git reset –-hard e2d5
- attention: only use it, when changes are not published (remotely) yet.
- → It is your command, IN CASE your are telling yourself, omg, what's that, what did i do here, let me undo that

Example

```
git reset --hard 2343
```

Docker

Install docker on Ubuntu

Walkthrough

```
sudo su -
apt update && apt install -y apt-transport-https ca-certificates curl software-
properties-common && curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key
add -;
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal
stable" && apt-get update && apt-cache policy docker-ce && apt-get install -y docker-
ce && systemctl status --no-pager docker
```

Important commands

Volume 1

```
mkdir testdir
cd testdir
```

```
## Dockerfile anlegen
docker build -t meincontainer .
docker images
## interactive starten
## nach exit wird er beendet
docker run -it meincontainer
## im container exit
docker ps -a
```

Volume 2

```
## image von docker hub download . hier ubuntu:latest
docker pull ubuntu

## Alle lokalen images anzeigen (auf dem Server vorhandene)
## z.B. die auf dem Serer mit docker build . erstellt wurden
## ohne downgeloaded von docker hub
docker images

## Neues docker container starten auf basis das ubuntu:latest images
## Im Hintergrund (Daemonized) und an ein Terminal
docker run -dt ubuntu:latest

## Alle laufenden docker-container anzeigen
docker ps

## Alle docker - container (auch beendete anzeigen)
docker ps -a

## Alle laufenden Container anzeigen
docker exec -it e1a1d3 bash

## Laufenden Docker container beendet und löschen
docker rm -f e21

## docker images anzeigen
docker images

## docker image lokal löschen
docker rmi ubuntu:latest
```

github pages

Github Pages

Types of Pages

- Personal Page: http://jmetzger.github.io
- Project Page http://

Personal Site

```
## Step 1: create personal repo
e.g.
https://github.com/gittrainereu/gittrainereu.github.io

git clone https://github.com/gittrainereu/gittrainereu.github.io
cd gittrainereu.github.io
echo "Hello World" > index.html
git add -A
git commit -m "Initial commit"
git push -u origin master

https://gittrainereu.github.io
```

Project Page

github actions

General overview

Komponenten

- actions
- workflows
- jobs
- steps
- events

Actions

- What can we do ?
- i.d.R läuft jede Action in einem docker-container (aber auch möglich: node-js (12), combined)

Workflows

- ENV (Umgebungsvariablen, Variablen)
- Jobs -> Steps
- Events

Add a self-host runner

Prerequisites

- Install docker

Walkthrough

```
1. Login to github
2. Click on repo -> settings
3. Click on actions
4. Click on runners
5. Click add self-hosted runner

## important, as we install it as root,
## you need to
export RUNNER_ALLOW_RUNASROOT="1"
## before doing the configuration ./config.sh
```

```
see:
https://serverfault.com/questions/1052695/must-not-run-with-sudo-while-trying-to-create-a-runner-using-github-actions

## When configuration is done, install service and start it.
./svc.sh install
./svc.sh start
./svc.sh status
```

Using it for an action:

```
## Example
## You need to activated it as:
## runs-on: [self-hosted, linux, x64, gpu]
## minimal would be
runs-on: self-hosted
```

Full example

- in .github/workflows/whatevername.yml

```
name: GitHub Actions Demo
on: [push]
jobs:
  Explore-GitHub-Actions:
    runs-on: self-hosted
    steps:
      - run: echo "🎉 The job was automatically triggered by a ${{ github.event_name }} event."
      - run: echo "🐧 This job is now running on a ${{ runner.os }} server hosted by GitHub!"
      - run: echo "🔎 The name of your branch is ${{ github.ref }} and your repository is ${{ github.repository }}."
      - name: Check out repository code
        uses: actions/checkout@v2
      - run: echo "💡 The ${{ github.repository }} repository has been cloned to the runner."
      - run: echo "🖥️ The workflow is now ready to test your code on the runner."
      - name: List files in the repository
        run: |
          ls ${{ github.workspace }}
      - run: echo "🍏 This job's status is ${{ job.status }}."
```

View logs of runner - service

```
systemctl status actions.runner.gittrainereu-runnertest.gh-runner1 -l
journalctl -u actions.runner.gittrainereu-runnertest.gh-runner1
```

Reference

- https://docs.github.com/en/actions/hosting-your-own-runners/adding-self-hosted-runners

Create dependant jobs

Execute job, if referred (by: needs) was succesful

```
name: Jochen's nicer workflow

on:
  # Triggers the workflow on push or pull request events but only for the master
branch
  push:
    branches: [ master ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
      - name: Run a one-line script
        run: |
          pwd
          ls -la
          /bin/false

  deploy:

    # needs a succesful build
    # THAT IS IMPORTANT
    needs: build
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
      - name: Starting the deploy
        run: |
          echo "starting the deployment process"
          ls -la
```

Ref:

- https://www.edwardthomson.com/blog/github_actions_17_dependent_jobs.html

Create custom composite action

Walkthrough

```
## new repo - e.g.
bash-action

## action.yml
name: 'Hello World'
description: 'Greet someone'
inputs:
```

```
  who-to-greet:  # id of input
    description: 'Who to greet'
    required: true
    default: 'World'
outputs:
  random-number:
    description: "Random number"
    value: ${{ steps.random-number-generator.outputs.random-id }}
runs:
  using: "composite"
  steps:
    - run: echo Hello ${{ inputs.who-to-greet }}.
      shell: bash
    - id: random-number-generator
      run: echo "::set-output name=random-id::$(echo $RANDOM)"
      shell: bash
    - run: ${{ github.action_path }}/goodbye.sh
      shell: bash

## goodbye.sh
echo "Goodbye"

### use it in other repo in workflow
## .github/workflows/workflow-hello.yml
on: [push]

jobs:
  greetings:
    runs-on: ubuntu-latest
    name: Greet Again

  steps:
    - id: foo
      uses: gittrainereu/bash-action@main
      with:
        who-to-greet: 'Mona the Octocat'
    - run: echo random-number ${{ steps.foo.outputs.random-number }}
      shell: bash
```

Type of actions

- JavaScript
- Docker
- Composite
- Ref: https://docs.github.com/en/actions/creating-actions/about-custom-actions#types-of-actions

Create a composite action

- https://docs.github.com/en/actions/creating-actions/creating-a-composite-action

Reference

- https://docs.github.com/en/actions/creating-actions

Create custom docker action

Walkthrough

```
##Dockerfile
## Container image that runs your code
FROM alpine:3.10

## Copies your code file from your action repository to the filesystem path `/` of the
container
COPY entrypoint.sh /entrypoint.sh

## Code file to execute when the docker container starts up (`entrypoint.sh`)
ENTRYPOINT ["/entrypoint.sh"]
```

```
## action.yml
name: 'Hello World'
description: 'Greet someone and record the time'
inputs:
  who-to-greet:  # id of input
    description: 'Who to greet'
    required: true
    default: 'World'
outputs:
  time: # id of output
    description: 'The time we greeted you'
runs:
  using: 'docker'
  image: 'Dockerfile'
  args:
    - ${{ inputs.who-to-greet }}
```

```
## entrypoint.sh
##!/bin/sh -l

echo "Hello $1"
time=$(date)
echo "::set-output name=time::$time"
```

```
## .github/workflows/workflow-docker.yml
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    name: A job to say hello
    steps:
      - name: Hello world action step
        id: hello
        uses: gittrainereu/docker-action@main
        with:
          who-to-greet: 'Mona the Octocat'
      # Use the output from the `hello` step
```

```
      - name: Get the output time
        run: echo "The time was ${{ steps.hello.outputs.time }}"
```

Reference:

- https://docs.github.com/en/actions/creating-actions/creating-a-docker-container-action

Work with artefacts

Walkthrough

```
name: Share data between jobs

on: [push]

jobs:
  job_1:
    name: Add 3 and 7
    runs-on: ubuntu-latest
    steps:
      - shell: bash
        run: |
          expr 3 + 7 > math-homework.txt
      - name: Upload math result for job 1
        uses: actions/upload-artifact@v2
        with:
          name: homework
          path: math-homework.txt

  job_2:
    name: Multiply by 9
    needs: job_1
    runs-on: windows-latest
    steps:
      - name: Download math result for job 1
        uses: actions/download-artifact@v2
        with:
          name: homework
      - shell: bash
        run: |
          value=`cat math-homework.txt`
          expr $value \* 9 > math-homework.txt
      - name: Upload math result for job 2
        uses: actions/upload-artifact@v2
        with:
          name: homework
          path: math-homework.txt

  job_3:
    name: Display results
    needs: job_2
    runs-on: macOS-latest
    steps:
```

```
      - name: Download math result for job 2
        uses: actions/download-artifact@v2
        with:
          name: homework
      - name: Print the final result
        shell: bash
        run: |
          value=`cat math-homework.txt`
          echo The result is $value
```

Reference

- https://docs.github.com/en/actions/advanced-guides/storing-workflow-data-as-artifacts

Create digitalocean-kubernetes.md

Walkthrough

```
## Step 1: Setup Kubernetes through digitalocean interface

## Step 2: Setup Container Registry (digitalocean)
(if not setup create a container registry)
ours is currently: training

## Step 3a: Create personal access key
https://cloud.digitalocean.com/account/api/

## Step 3b: ... and save it as secret DIGITALOCEAN_ACCESS_TOKEN in your repo
Repo -> Settings -> Secrects -> New Repository Secret (Button top left)

## Step 4: Kubernetes Cluster (digitalocean) mit Registry verheiraten (digitalocean)
In the control panel, you can select the Kubernetes clusters to use with your
registry.
This generates a secret, adds it to all the namespaces in the cluster and updates
the default service account to include the secret, allowing you to pull images from
the registry.

Container Registry -> Settings (Tab) -> Digital Ocean Kubernetes Integration -> Edit

Integrate all clusters -> Save (Button) (or only one specific cluster)
```

Reference

- https://docs.digitalocean.com/products/kubernetes/how-to/deploy-using-github-actions/

Deploy to server with ssh

Requirements

```
## apache is installed with php
## ssh runs
## DocumentRoot /var/www/html
```

Steps

```
Step 1:
=======
Auf Zielsystem (Webserver) public / private key erstellt
und pub-key in authorized_keys eingetragen.

cd /root/.ssh
## Achtung bitte rsa und 4096 nehmen, Beschreibung von github
## zum Erstellen eines pub/private keys funktioniert für github runner nicht
ssh-keygen -t rsa -b 4096 -C "foo@foo.com"
cat github-actions.pub >> authorized_keys
## Kopieren dieses Inhalt in die Secrets des repositories, von dem aus
## ihr deployen wollt
cat github-actions

Step 2: Eintrag in die Secretes
=======
## Repository -> Settings -> Secrets -> Actions -> New Secret for Repo
SSH_PRIVATE_KEY
## Hier dann der Wert von github-actions

## Host (IP Eintragen)
SSH_HOST

Step 3: Workflow einrichten in Repo unter
.github/workflows/deinworkflow.yml

## This is a basic workflow to help you get started with Actions

name: Jochen's nicer workflow

## Controls when the workflow will run
on:
  # Triggers the workflow on push or pull request events but only for the master
branch
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

## A workflow run is made up of one or more jobs that can run sequentially or in
parallel
jobs:
  # This workflow contains a single job called "build

  deploy:

    # needs a succesful build
```

```
    needs: build
    runs-on: ubuntu-latest

    env:
    # Beispiel, jedoch nicht notwendig
      SSH_PRIVATE_KEY: ${{ secrets.SSH_PRIVATE_KEY }}


    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
      - uses: actions/checkout@v2

      # Runs a single command using the runners shell
      - name: Starting the deploy
        run: |
          echo "starting the deployment process"
          ls -la
      - name: Install SSH Key
        uses: shimataro/ssh-key-action@v2
        with:
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          known_hosts: 'placeholder'
      - name: Adding Known Hosts
        run: ssh-keyscan -H ${{ secrets.SSH_HOST }} >> ~/.ssh/known_hosts
      - name: Show known hosts
        run: ls -la ~/.ssh/known_hosts
      - name: synchronize
        run: rsync -avz ./dist root@${{ secrets.SSH_HOST }}:/var/www/html/

### Schritt 3: Testen und debuggen
```

Ref:

- https://zellwk.com/blog/github-actions-deploy/

github actions - examples

Simple Workflow Test

```
## This is a basic workflow to help you get started with Actions
## .github/workflows/workflow-test.yml

name: Jochen's erster Workflow

## Controls when the workflow will run
on: push

## A workflow run is made up of one or more jobs that can run sequentially or in
parallel
jobs:
  # This workflow contains a single job called "build"
  jochen-runs-something:
```

```
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:

      # Runs a single command using the runners shell
      - run: echo Hello, world!
```

Checkout Repo

```
## This is a basic workflow to help you get started with Actions

name: Jochen's erster Workflow

## Controls when the workflow will run
on: push

## A workflow run is made up of one or more jobs that can run sequentially or in
parallel
jobs:
  # This workflow contains a single job called "build"
  jochen-checksout-and-runs-something:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:

      - name: Checke repo aus
        uses: actions/checkout@v2

      - run: |
          ls -la
          pwd
          env
      # Runs a single command using the runners shell
      - run: echo Hello, world!
```

Push to repo

```
## This is a basic workflow to help you get started with Actions

name: Jochen's erster Workflow

## Controls when the workflow will run
on: push

## A workflow run is made up of one or more jobs that can run sequentially or in
parallel
jobs:
```

```
  # This workflow contains a single job called "build"
  jochen-checksout-and-runs-something:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:

      - name: Checke repo aus
        uses: actions/checkout@v2

      - run: |
          ls -la
          pwd
          env
      # Runs a single command using the runners shell
      - run: echo Hello, world!
      - name: In repo schreiben
        run: |
          env > umgebung.txt
          ls -la >> umgebung.txt
          ls -la $GITHUB_WORKSPACE
          ls -la

      - name: Commit files
        run: |
          git config --local user.email "41898282+github-
actions[bot]@users.noreply.github.com"
          git config --local user.name "github-actions[bot]"
          git add .
          git commit -m "Add changes" -a

      - name: Push changes
        uses: ad-m/github-push-action@master
```

Nix kaputtmachen - so gehts

Die 5 goldenenen Regeln

```
1. Kein git commit --amend auf bereits veröffentlicht (gepushed) commit.

2. Kein git reset vor bereits veröffentlichte (gepushed/gepushten) commits
(1234 (HEAD -letzter Commit) < 5412 (vö - HEAD~1 - vorletzte Commit ) -> kein reset
auf 1234)

3. Mach niemals ein git push --force (JM sagt)

4. Kein Rebase auf bereits veröffentlichte commits (nach vö von Feature branchen)
- ausser Feature-Branch kann online gelöscht und nochmal erstellt werden
```

Tips & tricks

Beautified log

Walkthrough

```
git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset \
-%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset'"
```

PRETTY FORMATS

- all documented in git help log (section PRETTY FORMAT)
- https://git-scm.com/docs/git-log#_pretty_formats

Change already committed files and message

```
## Walkthrough
touch newfile.txt
git add .
git commit -am "new file added"

## Uups forgotten README
touch README
git add .
git commit --amend # README will be in same commit as newfile.txt
## + you can also changed the commit message
```

Best practice - Delete origin,tracking and local branch after pull request/merge request

```
## After a succesful merge or pull request und gitlab / github
## Follow these steps for a succesful cleanup

## 1. Delete feature branch in web interface (e.g. gitlab / github)
## e.g. feature/4811

## 2. Locally on your system prune the remote tracking branch
git fetch --prune

## 3. Switch to master or main (depending on what you master branch is)
git checkout master

## 4. Delete local branch
git branch -d feature/4811
```

Change language to german - Linux

```
sudo update-locale LANG=en_US.UTF-8
su - kurs

## back to german

sudo update-locale LANG=de_DE.UTF-8
su - kurs
```

```
## Reference:
https://www.thomas-krenn.com/de/wiki/Locales_unter_Ubuntu_konfigurieren

## update-locale does a change in
$ cat /etc/default/locale
LANG=en_US.UTF-8
```

Reference tree without sha-1

Always do pull --rebase for master branch

```
git config --global branch.master.rebase true
```

Exercises

merge feature/4712 - conflict

Exercise

```
1. You are in master-branch
2. Checkout new branch feature/4712
3. Change line1 in todo.txt
4. git add -A; git commit -am "feature-4712 done"
5. Change to master
6. Change line1 in todo.txt
7. git add -A; git commit -am "change line1 in todo.txt in master"
8. git merge feature/4712
```

merge request with bitbucket

```
## Local
git checkout -b feature/4822
ls -la
touch f1.txt
git add .
git commit -am "f1.txt"
touch f2.txt
git add .
git commit -am "f2.txt"
git push origin feature/4822
```

Online bitbucket

```
## create merge request
## and merge
```

Delete branch online after merge

Cleanup locally

```
git fetch --prune
git checkout master
git branch -D feature/4822
git pull --rebase
```

Snippets

publish lokal repo to server - bitbucket

```
# Step 1: Create repo on server without README and .gitignore /set both to NO when
creating

# Step 2: on commandline locally
cd /path/to/repo
git remote add origin https://erding2017@bitbucket.org/erding2017/git-remote-
jochen.git
git push -u origin master

# Step 3: for further commits
echo "test" > testdatei
git add .
git commit -am "added testdatei"
git push
```

failure-on-push-fix

```
## Step 1: push produces error
## you have done git push -u origin master the last to setup remote tracking branch by
option -u
git push
Password for 'https://erding2017@bitbucket.org':
To https://bitbucket.org/erding2017/git-remote-jochen.git
 ! [rejected](fetch first)
error: failed to push some refs to 'https://erding2017@bitbucket.org/erding2017/git-
remote-jochen.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
## Step 2: Integrate changes from online
git pull
## Step 2a: Editor opens and you need to save and ext (without changing anything)

## Step 3: re-push
git push
```

failure-on-push-with-conflict

Failure push

```
## Step 1: push produces error
## you have done git push -u origin master the last to setup remote tracking branch by
option -u
git push
Password for 'https://erding2017@bitbucket.org':
To https://bitbucket.org/erding2017/git-remote-jochen.git
 ! [rejected](fetch first)
....
## Step 2: Integrate changes from online
git pull

## Step 3: Solve conflict
Auto-merging agenda.txt
CONFLICT (content): Merge conflict in agenda.txt
Automatic merge failed; fix conflicts and then commit the result.
kurs@ubuntu-tr01:~/training$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

## Step 3a: Open file agenda.txt
## Decide for which version
## - remove all <<<<<< and ====== and >>>>>>>>>>  - lines

## Step 3b: then: save + exit from editor


## Step 3c: mark resolution
git status
git add todo.txt

## Step 3d:
git status
## as written there
git commit


## Step 4: re-push
git push
```

recipe

```
git push # failure
git pull
git add todo.txt
git commit
git push
```

Extras

Best practices

- Delete branches, not needed anymore
- git merge --no-ff -> for merging local branches (to get a good history from local)
- from online: git pull --rebase // clean history from online, not to many branches
- nur auf einem Arbeiten mit max. 2 Teilnehmern, wenn mehr feature-branch

Teil 2:

- Be careful with git commands that change history.
  - never change commits, that have already been pushed
- Choose workflow wisely
- Avoid git push -f in any case // should not be possible
- Disable possibility to push -f for branch or event repo

Using a mergetool to solve conflicts

Meld (Windows)

- https://meldmerge.org/

Configuration in Git for Windwos (git bash)

```
## you have to be in a git project
git config --global merge.tool meld
git config --global diff.tool meld
## Should be on Windows 10
git config --global mergetool.meld.path
“/c/Users/Admin/AppData/Local/Programs/Meld/Meld.exe”
## sometimes here
git config --global mergetool.meld.path "/c/Program Files (x86)/Meld/Meld.exe"

## do not create an .orig - file before merge
git config --global mergetool.keepBackup false
```

How to use it

```
## when you have conflict you can open the mergetool (graphical tool with )
git mergetool
```

Help

Help from commandline

On Windows

```
## on git bash enter
git help <command>
## e.g.
git help log

## --> a webpage will open with content
```

Documentation

GIT Pdf

- http://schulung.t3isp.de/documents/pdfs/git/git-training.pdf

GIT Book EN

- https://git-scm.com/book/en/v2

GIT Book DE

- https://git-scm.com/book/de/v2

Third Party Tools

Continuous Integration / Continous Deployment (CI/CD)

```
## Test often / Test automated (CI)

* Jenkins
* Github Actions
* Git Webhooks

## Publish new versions frequently (CD)

* Jenkins
* Github Action
* Git Webhooks
```