

gitops-Training

Agenda

1. Installation (GIT)

- [GIT auf Ubuntu/Debian installieren](#)
- [GIT unter Windows installieren](#)

2. Kubernetes

- [Installation micro8ks \(Ubuntu\)](#)

3. Commands (with tips & 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](#)
- [git rm](#)

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](#)
- [If example](#)
- [Work with artefacts](#)
- [Create digitalocean-kubernetes.md](#)
- [Deploy to server with ssh](#)

8. github actions - passing data

- [passing data from step to step](#)

9. github actions - events (IMHO trigger)

- [Events](#)
- [Required Status Checks](#)

10. github actions - examples

- [Simple Workflow Test](#)
- [Checkout Repo](#)
- [Push to repo](#)
- [Write secret to file and push to repo](#)

11. github actions - use case

- [Check lang-file before merging and disallow merging](#)
- [Run script from repo](#)
- [Deploy with ansible using ssh](#)

12. github - actions - docker

- [Was darf in das Dockerfile rein](#))

13. github - actions GITHUB_OUTPUT - GITHUB_SUMMARY

- [Write to summary page from within jobs](#)

14. github - actions - documentations

- [github actions repo](#)
- [github actions marketplace](#)
- [default environment variables](#)
- [Documentation github actions](#)

15. Nix kaputtmachen - so gehts

- [Die 5 goldenenen Regeln](#)

16. 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](#)

17. Exercises

- [merge feature/4712 - conflict](#)
- [merge request with bitbucket](#)

18. Snippets

- [publish lokal repo to server - bitbucket](#)
- [failure-on-push-fix](#)
- [failure-on-push-with-conflict](#)

19. Extras

- [Best practices](#)

- [Using a mergetool to solve conflicts](#)

20. Help

- [Help from commandline](#)

21. 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 tips & 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

* nonsense commit-message because of missing text-expertise"
## enter on last line
```

Change last commit-message (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 resolution 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
```

git rm

Deleting files only from repo (not locally)

```
git rm --cached filename.txt
## Please be sure to commit the change afterwards
## to reflect the changes in repo
git commit -am "my filename.txt was deleted"
```

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 Server 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

Events

- Events sind Ereignisse die stattfinden (man könnte auch sagen -> Trigger)
- Ref: <https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows>

Beispiele für Typen

- pull_request
 - if no activity types are specified, the workflow runs when a pull request is opened or reopened or when the head branch of the pull request is updated

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

Step 1: create new repo

```

## new repo - e.g. <tl>-bash-action
## z.B. fl-bash-action

```

Step 2: create action.yml in repo (toplevel)

```

## action.yml - im toplevel

```

```

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:
    - uses: actions/checkout@v4
    - run: echo Hello ${{ inputs.who-to-greet }}.
      shell: bash
    - id: random-number-generator
      run: echo "random-id=$RANDOM" >> $GITHUB_OUTPUT
      shell: bash
    - run: |

```

```
env
chmod u+x ${GITHUB_ACTION_PATH}/goodbye.sh
${GITHUB_ACTION_PATH}/goodbye.sh
shell: bash
```

Step 3: Create script

```
## goodbye.sh
echo "Goodbye"
```

Step 4: workflow erstellen

```
### 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 ${GITHUB_OUTPUT}
        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 "time=$time" >> $GITHUB_OUTPUT
```

```
## .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>

If example

```
steps:
  - name: Check for outdated packages
    id: vars
    run: |
      OUTDATED=$(npm outdated) || true

      echo "OUTDATED='$OUTDATED'" >> $GITHUB_OUTPUT

  - name: Upgrade
    if: ${ steps.vars.outputs.OUTDATED != '' }
    run: npm upgrade
```

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 -> Secrets -> 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

Deploying to server (without additional action)

Step 0: Setup Server with apache2 (Debian / Ubuntu)

```
apt install httpd
```

Step 1:

```
## Auf Zielsystem (Linux-Server Ubuntu/Debian) 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
## be nachfrage name key-> github-actions
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
```

Step 2.5: add files to repo in dist

```
##add file with content
dist/test.html
```

Step 3: Workflow

```
on:
  push:
    branches:
      - main
      - master
  workflow_dispatch:

jobs:
  run_pull:
    name: run pull
    # That is the image we are running on
    runs-on: ubuntu-latest

    steps:
```

```

- name: checkout
  uses: actions/checkout@v4
- name: install ssh keys
  # check this thread to understand why its needed:
  # https://stackoverflow.com/a/70447517
  run: |
    install -m 600 -D /dev/null ~/.ssh/id_rsa
    echo "${{ secrets.SSH_PRIVATE_KEY }}" > ~/.ssh/id_rsa
    ssh-keyscan -H "${{ secrets.SSH_HOST }}" > ~/.ssh/known_hosts
- name: connect and execute
  run: ssh "${{ secrets.SSH_USER }}"@"${{ secrets.SSH_HOST }}" "ls -la"
- name: infos
  run: |
    ls -la
    env
- name: synchronize
  run: rsync -avz dist/ root@"${{ secrets.SSH_HOST }}:/var/www/html/

- name: cleanup
  run: rm -rf ~/.ssh

```

Requirements (OLD VERSION from here)

```

## 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 - passing data

passing data from step to step

```
name: stepscheck

on:
  push:

jobs:
  job1:
    runs-on: ubuntu-latest
    steps:

      - name: Set color
        id: color-selector
        run: |
          echo $GITHUB_OUTPUT # test
          echo "SELECTED_COLOR=green" >> "$GITHUB_OUTPUT"
      - name: Get color
        env:
          SELECTED_COLOR: ${ steps.color-selector.outputs.SELECTED_COLOR }
        run: echo "The selected color is $SELECTED_COLOR"
```

github actions - events (IMHO trigger)

Events

```
9. events

9.1. Nur triggern, wenn bestimmte Dateien geändert wurden

on:
  pull_request:
    paths:
      - '**.js'

9.2 Branches ausschliessen
```

```

on:
  push:
    # Sequence of patterns matched against refs/heads
    branches-ignore:
      - 'mona/octocat'
      - 'releases/**-alpha'
    # Sequence of patterns matched against refs/tags
    tags-ignore:
      - v2
      - v1.*

```

Refs:

- https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows#pull_request
- <https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows#push>

Required Status Checks

You can find this setting under your repository's **Settings** > **Branches**.

☒ **Require status checks to pass before merging**
 Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☐ **Require branches to be up to date before merging**
 This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).

Status checks found in the last week for this repository

<input type="checkbox"/> .github/workflows/link-checker-on-pr.yml	
<input checked="" type="checkbox"/> .github/workflows/on-pull-request.yml	Required

There are further details about these settings in the documentation.

<https://help.github.com/en/github/administering-a-repository/enabling-required-status-checks>

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

```

Write secret to file and push to repo

```

name: secret and push

on: [push]

jobs:
  job1:
    runs-on: ubuntu-latest
    permissions:
      # Job-level permissions configuration starts here

```



```

    contents: write          # 'write' access to repository contents
    pull-requests: write    # 'write' access to pull requests
steps:
  - uses: actions/checkout@v4
    with:
      fetch-depth: 0 # otherwise, there would be errors pushing refs to the
destination repository.
  - name: Create local changes 1
    run: |
      touch somefile.txt
  - name: Create local secret
    shell: bash
    env:
      SUPER_SECRET: ${ secrets.SECRET_SUPER_SECRET }
    run: |
      echo "$SUPER_SECRET"
      echo "foo" > myfile
      echo
      echo "OUTPUT 1: -----"
      cat myfile
      echo "EOF OUTPUT1"
      echo
      echo "OUTPUT 2: -----"
      echo "$SUPER_SECRET" >> myfile
      cat myfile
      echo "EOF OUTPUT 2"
      echo
      echo "OUTPUT 3: -----"
      echo "foo2" >> myfile
      cat myfile
      echo "EOF OUTPUT 3"
  - 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 -A
      git commit -a -m "Add changes"
  - name: Push changes
    uses: ad-m/github-push-action@master
    with:
      github_token: ${ secrets.GITHUB_TOKEN }
      branch: ${ github.ref }

```

github actions - use case

Check lang-file before merging and disallow merging

Step 1: Prerequisites (in master)

```
## only or locally
## create files in master
dist/index.html
dist/en/index.html
```

```
## workflow yaml
.github/workflows/lang.yaml
```

```
name: langchecker

on:
  push:
    branches:
      - 'feature/**'
    paths:
      - 'dist/index.html'

jobs:
  translation-check:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0 # otherwise, there would be errors pushing refs to the
destination repository.
      - run: |
          ls -la
          git branch
          git branch -r
          # grep always returns 1 by design if does not find a result
          echo "--HEADER--" > ANALYZER
          git diff --name-only HEAD origin/master -- dist/en/index.html >> ANALYZER
          COUNT=$(cat ANALYZER | grep -cE "(--HEADER--|dist/en/index.html)")
          # 1 - Only HEADER line will be present
          if [ $COUNT -eq 1 ] ; then echo "dist/en/index.html not changed"; exit 1;
          else echo "Change in lang detected - Happy"; fi
```

```
git add -A; git commit -am "lang files"; git push
```

Step 2: create feature

```
''' git checkout -b feature/6 '''
```

```
## change
dist/index.html
```

```
git add -A; git commit -am "new version2"
git push -u origin feature/6
```

Workflow müsste jetzt unter actions triggern und fehler werden, weil dist/en/index.html (englische Version) nicht geändert wurde

Run script from repo

Step 1: Create script in repo

```
## scripts/deploy.sh
```

```
#!/bin/bash

echo "test this"
env
echo $GITHUB_OUTPUT
echo "VORNAME=Hans" >> $GITHUB_OUTPUT
```

Step 2: .github/workflows/blank.yml

```
name: CI

## 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" ]

  # 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:
  run-playbooks:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: run deploy
        shell: bash
        run: |
          cd scripts
          chmod u+x ./deploy.sh
          ./deploy.sh
```

Deploy with ansible using ssh

Create files

```
## infrastructure/ansible/setup-prod.yml
```

```

---
- hosts: all
  tasks:
    - name: install packages
      become: true
      become_user: root
      apt:
        state: present
        name:
          - htop

```

```

## infrastructure/ansible/hosts
## anpassen mit deinem host un der ip (Trainer fragen ;o))

```

```

gr1.t3isp.de ansible_host=167.172.179.197

```

Create workflow

```

name: CI

## 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" ]

  # 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:
  run-playbooks:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup SSH
        shell: bash
        run: |
          eval `ssh-agent -s`
          mkdir -p /home/runner/.ssh/
          touch /home/runner/.ssh/id_rsa
          echo -e "${{secrets.SSH_PRIVATE_KEY}}" > /home/runner/.ssh/id_rsa
          chmod 700 /home/runner/.ssh/id_rsa
          ssh-keyscan -t rsa,dsa,ecdsa,ed25519 ${{secrets.SSH_HOST}} >>
/home/runner/.ssh/known_hosts
      - name: Run ansible script
        shell: bash
        run: |

```

```
cd infrastructure/ansible
cat setup-prod.yml
ansible-playbook -vvv --private-key /home/runner/.ssh/id_rsa -u
${{secrets.SSH_USER}} -i hosts setup-prod.yml
```

github - actions - docker

Was darf in das Dockerfile rein

- <https://docs.github.com/de/actions/creating-actions/dockerfile-support-for-github-action>

github - actions GITHUB_OUTRPUT - GITHUB_SUMMARY

Write to summary page from within jobs

- Writing to \$GITHUB_STEP_SUMMARY writes to a summary, that is visible on the summary of the actions - run

```
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: |
          echo "### Hello world! :rocket:" >> $GITHUB_STEP_SUMMARY
          pwd
          ls -la
          #/bin/false
          echo "### Hello world in build after false ! :rocket:" >>
          $GITHUB_STEP_SUMMARY

  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
echo "### Hello world in deploy after false ! :rocket:" >>
$GITHUB_STEP_SUMMARY
```

github - actions - documentations

github actions repo

- <https://github.com/actions/checkout>

github actions marketplace

- <https://github.com/marketplace?category=&query=&type=actions&verification=>

default environment variables

- <https://docs.github.com/en/actions/learn-github-actions/variables#default-environment-variables>

Documentation github actions

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

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"

## Ups 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]          master -> master (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]          master -> master (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/>

Find out if mergetool meld is available

```
## Important: close and reopen git bash before doing that
## you can try to see, if meld can be executed by simply typing "meld"

git mergetool --tool-help
```

Configure, when it is found by mergetool --tool-help

```
## you have to be in a git project
git config --global merge.tool meld
git config --global diff.tool meld
git config --global mergetool.keepBackup false
git config --list
```

If not found bei mergetool --tool-help :: Configuration in Git for Windows (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 / Continuous Deployment (CI/CD)

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

* Jenkins
* Github Actions
* Git Webhooks

## Publish new versions frequently (CD)

* Jenkins
* Github Action
* Git Webhooks
```