GIT-Bitbucket-Jenkins-Training

Agenda

1. Geschichte / Grundlagen
   - GIT Pdf
2. Commands (with tipps & tricks)
   - git add + Tipps & Tricks
   - git alias with multiple commands
   - git commit
   - git log
   - git config
   - git show
   - Needed commands for starters
   - git branch
   - git checkout
   - git merge
   - git tag
   - git rm (Dateien löschen aus git)
3. Tips & tricks
   - Beautified log
   - Change already committed files and message
   - Best practice - Delete origin,tracking and local branch after pull request/merge request
   - Einzelne Datei auschecken
   - Always rebase on pull - setting
   - Arbeit mit submodules
   - Integration von Änderungen (commits, einzelne Dateien) aus anderen commits in den Master
   - Fix conflict you have in merge-request (gitlab)
   - SETUP.sql zu setup.sql in Windows (Groß- und Kleinschreibung)
   - Force specfic commit message
   - Alle Dateien, die sich geändert haben anzeigen z.B. heute
4. Exercises
   - merge feature/4712 - conflict
   - merge request with bitbucket
   - Exercise with cherry-picking
5. Snippets
   - publish lokal repo to server - bitbucket
   - failure-on-push-fix
   - failure-on-push-with-conflict
6. Extras
   - Best practices
   - Using a mergetool to solve conflicts
   - Overview GIT-Servers
   - 4 goldene Regeln
7. Help
   - Help from commandline
8. subtrees / submodules
   - substrees
   - submodules
9. Authentication
   - Work with different credentials
10. Documentation
    - GIT Pdf

  - GIT Book EN
  - GIT Book DE
  - GIT Book - submodules
  - GIT Guis
  - Third Party Tools
  - Specification Conventional Commits
  - https://www.innoq.com/de/talks/2019/05/commit-message-101/
  - https://github.com/GitAlias/gitalias/blob/main/gitalias.txt
  - https://education.github.com/git-cheat-sheet-education.pdf

11. Jenkins - Installation

    - Installation of Jenkins Server (Controller)
    - Installation of Linux - Agent - Docker

12. Jenkins - Backup

    - Backup / Restore

13. Jenkins - Pipelines

    - Declarative vs. Scripted

14. Jenkins - Declarative Pipeline

    - Comments
    - Working with shell-commands
    - Work with environment variables
    - Credentials in Umgebungsvariablen anzeigen
    - Run on docker agent
    - Cleanup after pipeline run/job
    - Change variable within stage in pipeline

Backlog

1. Installation

   - GIT auf Ubuntu/Debian installieren
   - GIT unter Windows installieren

2. Erweiterte Commands

   - git reflog
   - git reset - Back in Time

GIT Pdf

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

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 alias with multiple commands

Multiple commands in one alias

```
git config --global alias.ac '!git add . && git commit -am'
```

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

Checkout (change to) existing branch

```
git checkout feature/4711
```

Checkout and create branch

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

File aus einem Commit holen (oder HEAD)

```
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 tags, Working with 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
```

git delete tag

```


## 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
```

Misc

```
## 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
```

git rm (Dateien löschen aus git)

Datei nur aus Repo und Index löschen

```
git rm --cached dateiname
```

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

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
```

Einzelne Datei auschecken

aus anderem Commit

```
## aus commit 11ed

git checkout 11ed -- todo.txt
## unterverzeichnis
git checkout 11ed -- tmp/test.txt

```

...und direkt umbenennen

```
## datei todo.txt aus 11ae -> Inhalt anzeigen und direkt neue datei umleiten
git show 11ae:todo.txt > todoneu.txt

## ein commit vorher
git show 11ae^:todo.txt > todoneu.txt
```

Always rebase on pull - setting

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

Arbeit mit submodules

Best practive

```
clone repo use for submodule seperately
(in seperate folder)
if you want to change it
```

Updating commands for updating subfolder

```
git submodule update --remote
## use other branch from submodule then master
git config -f .gitmodules submodule.DbConnector.branch stable
```

Ref.

- https://git-scm.com/book/de/v2/Git-Tools-Submodule

Integration von Änderungen (commits, einzelne Dateien) aus anderen commits in den Master

Walkthrough

```
## 1. Schritt - erstellen integrationsbranch von dev/staging branch
git checkout -b integrate/1

## Möglichkeit 1: cherry-pick - komplette commit inkl. aller Änderungen mit reinnehmen
## Hier wird gemerged: Gemerged
## Evtl. Konflikt, den muss ich dann lösen
git cherry-pick c5906c0

## Möglichkeit 2: Einzelne files aus commit: Achtung, wenn im Work-Directory
## bereits vorhanden überschrieben
## commit wird bereits durchgeführt
git checkout ddb0 -- armin3.txt

## Möglichkeit 3: cherry-pick ohne commit
git cherry-pick -n 4497
git status
## alle files rausnehmen, die wir nicht haben möchten, wie folgt.
git restore --staged agenda.txt
## Achtung, jetzt sind diese so im Working Directory als unstaged
## d.h. die alte Version aus dem letzten Commit holen
git checkout HEAD -- agenda.txt

## 3. Schritt
## änderungen commiten
git commit -am "Revised version"

## 4. Nach online pushed
git push -u origin integrate/1

## 5. Merge request in gitlab: integrate/1 -> master
## und dann mergen online
```

Fix conflict you have in merge-request (gitlab)

Walkthrough

```
## create feature-branch and worked on it
git checkout -b feautre/4711
## ... changes
git add .; git commit -am "new feature"
## pushed branch online
git push -u origin feature/4711
## then created merge online
## feature/4711 --> master

###### TaDa - It was NOT possible to merge because of conflict
## unfortunately advice on gitlab/bitbucket is not worth the dime
## locally, update you feature-branch like so
## NO git pull --rebase please, otherwice, you have to redo you merge_request afterwards
## get changes from master
git pull origin master

## fix conflicts
git add .
git commit

## push new version of feature - branch online
git push

## now you can merge in the merge-request interface on gitlab
```

SETUP.sql zu setup.sql in Windows (Groß- und Kleinschreibung)

Problem

- Windows erkennt in git keine Änderung der Groß- und Kleinschreibung
- Workaround: git rm --cached; git commit -am

Walkthrough

```
touch SETUP.sql
git add .; git commit -am "SETUP neu"

## Uups, verschrieben ! Was jetzt ?
git rm --cached SETUP.sql # Datei wird aus git rausgenommen
git commit -am "und dingfest machen"
## Beweis
git show HEAD # letztes commit mit Änderungen anzeigen


## Jetzt auf ein Neues
## oder im Explorer
mv SETUP.sql setup.sql
git add .; git commit -am "setup.sql neu"
git show HEAD
```

Force specfic commit message

Basics

- Done on Server-Side
- Specific to server - Software (like github/gitlab)

Example - pre-receive-hook

- https://git-scm.com/book/en/v2/Customizing-Git-An-Example-Git-Enforced-Policy

Ref:

- https://docs.gitlab.com/ee/user/project/repository/push_rules.html (not free)
- https://docs.gitlab.com/ee/administration/server_hooks.html

Alle Dateien, die sich geändert haben anzeigen z.B. heute

Files

git log --after="2015-11-05T16:36:00-02:00" --before="2022-09-28" --pretty=format:"" --name-only | sort -u

Mit loop

```
for i in $(git log --after="2022-09-26" --before="2022-09-27" --pretty=format:"" --name-only | sort -u); do
git log -- $i; done
```

Änderungen einer datei

```
git log --after="2022-09-26" --before="2022-09-27" --pretty=format:"" --follow -p -- todo.txt
```

merge feature/4712 - conflict

Exercise

```
1. You are in master-branch
2. Checkout new branch feature/4714
3. Change name: in playbook.yml
4. git add -A; git commit -am "feature-4714 done"
5. Change to master
6. Change name in playbook.yml in todo.txt
7. git add -A; git commit -am "change name in playbook.yml in master"
8. git merge feature/4714
```

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 -u origin feature/4822
```

Online bitbucket / gitlab

```
## 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
```

Exercise with cherry-picking

Walkthrough

```
1. Neuen Branch feature/5050 erstellen
2. 3 Änderungen wie folgt:
   a. todo.txt Zeile1 + add -A + commit
   b. todo.txt Zeile2 + add -A + commit
   c. todo.txt Zeile3 + add -A + commit
3. Wechsel in den master
---
4. commit von 2b. notieren
5. branch löschen
6. Cherry-picken von commit aus 2b
```

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
```

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) - Install

- 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/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
```

Overview GIT-Servers

Builtin with git-installation

Simple GIT-Server

```
## included in installation with git
Cons: Can do nearly nothing (only pushing and pulling)

* no graphical interface
* no multi-user support
* no additional features (like bugtracking / milestones a.s.o)
```

Web-Interface (also from git installation)

```
Cons: Mo multi-user interaction
```

Comfortable Git-Server

gitea / codeberg

- OpenSource
- minimum feature
- not integrated with other software

gitlab

General

- On premise / cloud

Pros

- Devops - Server (Integration)
- Tools für Devops
- Integration von CI/CD
  - Favourite von Jochen (in opposite github actions)
- kleine Teams können on premise kostenlos starten
- Im Rahmen von DevOps auch automatische Integration von Scannen von Software drin.

bitbucket

Overview

- Software Company Atlassian.
- Problematic license policy
- Cloud-Based (SaaS) - ich miete - subscription
- On Premise (Installation im Firmennetz)
  - aber abgekündigt
- On Premise fü+r grosse Unternehmen - sehr teuer

Pros

- Integration with other software products (confluence - wiki, jira - ticket system)
- webhooks (url aufgerufen wird dich ich festlege mit einem payload)

Cons

- No CI/CD directly within bitbucket

github

Overview

- Bought by microsoft

Pros

- on premise git gut möglich (github enterprise)
- Editor sehr gut im Web-Interface

Cons

- Menüführung von github nicht so intuitiv für Jochen

- github actions (CI/CD) zu kompiziert (Lernkurve größe als bei gitlab ci/cd)

Azure Devops

Overview

- Repos are use from github under the hood

Con

- Lernkurve höher als bei github, gitlab, bitbucket

Pros

- Sicherheitsfeatures höher
- Integration mit VisualStudio
- Kostenvorteile durch Lizenz Visual Studio Pro

AWS Code Commit

Overview

- Innerhalb der Amazon AWS Familie

Pros

- Integration von AWS

Cons

- Etwas ungünstige Positionierung des Interface (wo finde ich das überhaupt)
- Benamung: AWS Console -> Web Interface
- Sehr kleines FeatureSet (z.B. GIT LFS möglich)
- keinen Forken möglich

4 goldene Regeln

```
  * Niemals einen push --force machen
    (nur in Abstimmung mit dem gesamten Team)
  * kein reset vor bereits veröffentlichte commits
  * git commit --amend nur wenn commit noch nicht veröffentlicht (push auf server)
  * rebase nur wenn branch / commit noch nicht veröffentlicht
```

Help from commandline

On Windows

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

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

substrees

Prerequisites - Existing local repo

```
## in der bash
cd ..
cp -a training training-neu
cd training-neu
```

Walkthrough

```
git remote add -f training-git https://github.com/jmetzger/training-git.git
## weird, but needed
git status
git subtree add --prefix training training-git main --squash
```

Updating

```
git fetch training-git main
git subtree pull --prefix training training-git main --squash
```

Push

```
git subtree push --prefix=training training-git main
```

Ref.

- https://www.atlassian.com/git/tutorials/git-subtree

submodules

Best practive

```
clone repo use for submodule seperately
(in seperate folder)
if you want to change it
```

Updating commands for updating subfolder

```
git submodule update --remote
## use other branch from submodule then master
git config -f .gitmodules submodule.DbConnector.branch stable
```

Ref.

- https://git-scm.com/book/de/v2/Git-Tools-Submodule

Work with different credentials

Ref:

https://de.linkedin.com/pulse/mehrere-gitlabgithub-accounts-bzw-ssh-keys-zum-host-mit-mindermann

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

GIT Book - submodules

- https://git-scm.com/book/de/v2/Git-Tools-Submodule

GIT Guis

- https://git-scm.com/downloads/guis/

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
```

Specification Conventional Commits

- https://www.conventionalcommits.org/en/v1.0.0/

Installation of Jenkins Server (Controller)

Installation of Linux - Agent - Docker

Step 1: Create new machine (virtual machine)

```
##!/bin/bash

##### This need to be run as root

groupadd sshadmin
USERS="11trainingdo"
echo $USERS
for USER in $USERS
do
  echo "Adding user $USER"
  useradd -s /bin/bash --create-home $USER
  usermod -aG sshadmin $USER
  echo "$USER:11dortmund22" | chpasswd
done

## We can sudo with 11trainingdo
usermod -aG sudo 11trainingdo

## 20.04 and 22.04 this will be in the subfolder
if [ -f /etc/ssh/sshd_config.d/50-cloud-init.conf ]
then
  sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config.d/50-cloud-init.conf
fi

### both is needed
sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config

usermod -aG sshadmin root

## TBD - Delete AllowUsers Entries with sed
## otherwice we cannot login by group

echo "AllowGroups sshadmin" >> /etc/ssh/sshd_config
systemctl reload sshd

#### Now the docker / jenkins part

apt-get update
apt install -y --no-install-recommends openjdk-17-jdk-headless

## adding jenkins user
useradd -m -s /bin/bash jenkins

## needed for installation of agent.jar
apt-get install -y ca-certificates curl gnupg lsb-release

mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

apt-get update
apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

## groupadd docker
usermod -aG docker jenkins
usermod -aG sudo jenkins
```

```
### Alreaday install the service, although we need to do some manual steps
mkdir -p /usr/local/jenkins-service
chown jenkins /usr/local/jenkins-service

cat << EOF > /etc/systemd/system/jenkins-agent.service

[Unit]
Description=Jenkins Agent

[Service]
User=jenkins
WorkingDirectory=/home/jenkins
ExecStart=/bin/bash /usr/local/jenkins-service/start-agent.sh
Restart=always

[Install]
WantedBy=multi-user.target

EOF


## not sure, if daemon-reload is really needed on ubuntu 22.04 LTS
systemctl daemon-reload
systemctl enable jenkins-agent
```

Step 2: neuen Agent in jenkins anlegen

- http://164.90.173.76:8080/computer/

Step 3: java-Client in install-agent.sh /usr/local/jenkins-agent installieren und rechte setzen

```
## Achtung ip und anderen schlüssel
## also root
cd /usr/local/jenkins-service
vi start-agent.sh
```

```
## Example you get from you jenkins
## under new node
```

```
##!/bin/bash
cd /usr/local/jenkins-service
## Just in case we would have upgraded the controller, we need to make sure that the agent is using the
latest version of the agent.jar
curl -sO http://my_ip:8080/jnlpJars/agent.jar
java -jar agent.jar -jnlpUrl
http://my_ip:8080/computer/My%20New%20Ubuntu%2022%2E04%20Node%20with%20Java%20and%20Docker%20installed/jenkins-
agent.jnlp -secret my_secret -workDir "/home/jenkins"
exit 0
```

```
chmod u+x start-agent.sh
chown -R jenkins:jenkins /usr/local/jenkins-service
chown -R jenkins:jenkins /home/jenkins
systemctl start jenkins-agent
systemctl status jenkins-agent
```

- https://www.jenkins.io/blog/2022/12/27/run-jenkins-agent-as-a-service/

Backup / Restore

```
- Es gibt keine Datenbanken
- Es ist alles in files
- Heimatverzeichnis vom Controller sichern
(in paar Verzeichnisse kann man dort ausschliessen)
Key zur Verschlüsselung passwörter sollte getrennt aufbewahrt werden
```

```
Details: https://www.jenkins.io/doc/book/system-administration/backing-up/
```

Declarative vs. Scripted

Comments

```
// comment
```

Working with shell-commands

Example 1 - single lines sh

```
pipeline {
    agent any

    stages {
        stage('Hello') {
            steps {
                echo 'Hello World'
                echo 'good good'
                sh 'pwd'
                sh 'ls -la'
                sh 'ls -la > testx.txt'
                sh 'ls -la testx.txt'
                sh 'cat testx.txt'
            }
        }
    }
}
```

Example 2 - multiline sh

```
pipeline {
    agent any

    stages {
        stage('Hello') {
            steps {
                echo 'Hello World'
                echo 'good good'
                sh'''
                  pwd
                  ls -la
                  ls -la > test3.txt
                  ls -la test2.txt
                  cat test2.txt
                '''
            }
        }
    }
}
```

Work with environment variables

```
pipeline {
    agent any
    // for all stages
    environment {
        CC = 'clang'
    }
    stages {

        stage('Example') {
            // for a specific stage
            environment {
                MY_STAGE_ENV_VAR = 'Stage env var is this'
            }
            steps {
                sh 'printenv'
            }
        }

        stage('Only toplevel stage'){
            steps {
                sh 'printenv'
                sh 'env'
            }

        }

    }
}
```

Credentials in Umgebungsvariablen anzeigen

```
pipeline {
  agent any

  environment {
    DOCKER=credentials('docker-login')
  }

  stages {
     stage ('build'){
       steps {
         echo "$DOCKER_USR"
         echo "$DOCKER_PSW"
         sh 'echo hallo pass: $USER_PSW'
         sh 'echo hallo usr: $USER_USR'
         sh 'env'

         echo "${DOCKER_USR}"
         echo "${DOCKER_PSW}"

         echo  "${env.DOCKER_USR}"

         echo "$DOCKER_USR"

       }



     }

  }

}
```

Run on docker agent

Running docker / docker needs to be installed on agent

```
pipeline {
    agent {
        docker { image 'node:16.13.1-alpine' }
    }
    stages {
        stage('Test') {
            steps {
                sh 'node --version'

            }
        }
    }
}
```

Run on specific docker agents

```
pipeline {
    agent {
        docker {
            image 'node:16.13.1-alpine'
            label 'linux'

        }
    }
    stages {
        stage('Test') {
            steps {
                sh 'node --version'

            }
        }
    }
}
```

Using different docker images in different stages

```
pipeline {
    agent none
    stages {
        stage('Back-end') {
            agent {
                docker { image 'maven:3.9.0-eclipse-temurin-11' }
            }
            steps {
                sh 'mvn --version'
            }
        }
        stage('Front-end') {
            agent {
                docker { image 'node:16.13.1-alpine' }
            }
            steps {
                sh 'node --version'
            }
        }
    }
}
```

Cleanup after pipeline run/job

```
pipeline {
    agent any

    stages {
        stage('Hello') {
            steps {
                echo 'Hello World'
                echo 'good good'
                sh 'pwd'
                sh 'ls -la'
                sh 'ls -la > test.txt'
                sh 'ls -la test.txt'
                sh 'cat test.txt'
            }
        }
    }
    post {
        // Clean after build
        always {
            cleanWs(cleanWhenNotBuilt: false,
                    deleteDirs: true,
                    disableDeferredWipeout: true,
                    notFailBuild: true,
                    patterns: [[pattern: '.gitignore', type: 'INCLUDE'],
                               [pattern: '.propsfile', type: 'EXCLUDE'],
                               [pattern: 'test.txt', type: 'INCLUDE']])
        }
    }

}
```

Change variable within stage in pipeline

```
def animal="cat"
pipeline {
  agent any
  stages {
    stage("hello") {
      steps {
        script {
          echo animal
          animal = "dog";
        }
      }
    }
    stage("goodbye") {
      steps {
        script {
          echo animal
        }
      }
    }
  }
}
```

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

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
```