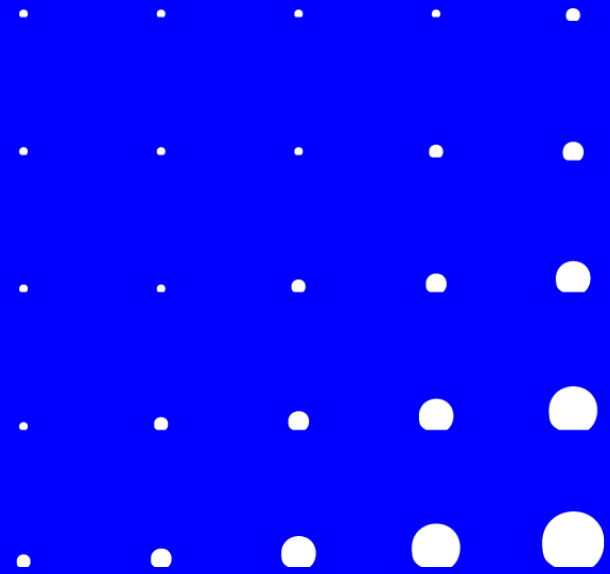
Technische Hochschule
mannheim

Versionskontrolle & Git-Workshop



Ablauf des Workshops

Technische Hochschule mannheim

Teil A – Theorie (kurz)

Was ist Versionskontrolle? Zentral vs. verteilt, Git-Grundkonzepte

Teil B – Workshop (Hands-on)

Setup & erstes Repository

Änderungen verwalten und rückgängig machen

Branching, Merging & Konflikte

Remotes & Zusammenarbeit (inkl. GitHub)

Workflows & Best Practices

Format

Erklärte Befehle im Wechsel mit kleinen Übungen zum Mitmachen

Lernziele

Technische Hochschule mannheim

Verstehen, warum Versionskontrolle unverzichtbar ist

Die Git-Grundkonzepte und den Datenfluss erklären können

Ein Repository anlegen, Änderungen committen und nachvollziehen

Mit Branches parallel arbeiten und Merges/Konflikte beherrschen

Mit einem Remote (z. B. GitHub) zusammenarbeiten

Sinnvolle Workflows und Best Practices anwenden

Technische Hochschule
mannheim

A.1 Grundlagen der Versionskontrolle

Warum wir ein VCS brauchen

Was ist Versionskontrolle?

Technische Hochschule mannheim

Definition

Ein Versionskontrollsystem (VCS) erfasst Änderungen an Dateien über die Zeit, sodass man jederzeit zu früheren Ständen zurückkehren kann

Was es festhält

Wer hat was wann und warum geändert?

Typische Anwendung

Quellcode – aber auch Doku, Konfiguration, Skripte, Texte

Kurz gesagt

Eine vollständige, nachvollziehbare Historie des Projekts

Warum Versionskontrolle? – Probleme ohne VCS

Technische Hochschule mannheim

Datei-Chaos

„bericht_final_v2_wirklich_final.docx“ – welche Version gilt?

Kein Verlauf

Was wurde wann geändert – und wie komme ich zurück?

Zusammenarbeit

Parallele Änderungen überschreiben sich gegenseitig

Kein sicheres Experimentieren

Angst, etwas kaputtzumachen, bremst Änderungen

Zentrale vs. verteilte Versionskontrolle

1	2	3
Lokal	**Zentral (CVCS)**	**Verteilt (DVCS)**
Nur eigener Rechner	Ein Server (z. B. SVN)	Git
z. B. Kopien/Ordner	Alle holen von dort	Jeder hat volle Historie
Kein Team	Online nötig	Offline arbeiten
Fehleranfällig	Single Point of Failure	Robust & schnell

Warum Git?

Technische Hochschule mannheim

Verteilt

Jede Arbeitskopie ist ein vollständiges Repository mit gesamter Historie

Schnell & effizient

Die meisten Operationen laufen lokal – kein Netzwerk nötig

Leichtgewichtige Branches

Verzweigen und Zusammenführen sind günstig und alltäglich

Datenintegrität

Inhalte werden über Prüfsummen (SHA-1/-256) abgesichert

Verbreitung

De-facto-Standard; entwickelt 2005 von Linus Torvalds für den Linux-Kernel

Technische Hochschule
mannheim

A.2 Git-Grundkonzepte

Das Modell hinter Git

Repository, Commit & Snapshot

Technische Hochschule mannheim

Repository (Repo)

Projektordner samt vollständiger Historie – liegt im versteckten Ordner .git

Commit

Ein gespeicherter Stand mit eindeutiger ID (Hash), Autor, Zeit und Nachricht

Snapshots statt Diffs

Git speichert den Zustand aller Dateien je Commit, nicht nur Zeilenänderungen

HEAD

Zeiger auf den aktuell ausgecheckten Commit bzw. Branch

Technische Hochschule
mannheim

Die drei Bereiche in Git

1

Working Directory

Deine Arbeitskopie

Hier wird editiert

Dateien: untracked

oder modified

2

Staging Area (Index)

Vormerkbereich

git add

Auswahl für den

nächsten Commit

3

Repository (.git)

Dauerhafte Historie

git commit

Unveränderliche

Snapshots

Der Lebenszyklus einer Datei

Technische Hochschule mannheim

untracked

Neu, von Git noch nicht erfasst

modified

Erfasst und seit dem letzten Commit geändert

staged

Für den nächsten Commit vorgemerkt (nach git add)

committed / unmodified

Sicher in der Historie gespeichert (nach git commit)

Merksatz

edit → add → commit – und das immer wieder

Technische Hochschule
mannheim

B.1 Setup & erstes Repository

Git einrichten und loslegen

Installation & Konfiguration

Technische Hochschule mannheim

Installation prüfen

```
git --version
```

Identität setzen (einmalig, global)

```
git config --global user.name "Max Mustermann"
git config --global user.email "max@example.com"
```

Standard-Branchname & Editor (optional)

```
git config --global init.defaultBranch main
```

Einstellungen ansehen

```
git config --list
```

Übung · Setup prüfen

Technische Hochschule mannheim

Aufgabe

Prüfe, ob Git installiert ist, und hinterlege deinen Namen und deine E-Mail.

Befehle

```
git --version
git config --global user.name "Dein Name"
git config --global user.email "du@example.com"
git config --list
```

Ergebnis

Name und E-Mail erscheinen in der Ausgabe von git config --list.

Repository anlegen: zwei Wege

1

Neu: init

Leeres Repo im

aktuellen Ordner

git init

erzeugt .git/

2

Vorhanden: clone

Kopie eines

Remote-Repos

git clone <url>

inkl. Historie

3

Danach

git status

zeigt den Zustand

der Arbeitskopie

Übung · Erstes Repository

Technische Hochschule mannheim

Aufgabe

Lege einen Projektordner an, initialisiere ein Git-Repository und sieh dir den Status an.

Befehle

```
mkdir mein-projekt
cd mein-projekt
git init
git status
```

Ergebnis

Git meldet: „On branch main, No commits yet“ und ein leeres Repository.

Der erste Commit

Technische Hochschule mannheim

Datei erstellen und Status prüfen

```
git status  # zeigt untracked files
```

Zum Commit vormerken (stagen)

```
git add README.md   # gezielt
git add .           # alle Änderungen
```

Commit erstellen

```
git commit -m "Initialer Commit: README hinzugefügt"
```

Historie ansehen

```
git log --oneline
```

.gitignore – was nicht ins Repo gehört

Technische Hochschule mannheim

Zweck

Generierte oder sensible Dateien von der Versionierung ausschließen

Typische Einträge

```
*.log
node_modules/
build/  target/  dist/
.env          # Secrets niemals committen!
```

Merke

Bereits getrackte Dateien ignoriert .gitignore nicht nachträglich

Übung · Commits erstellen

Technische Hochschule mannheim

Aufgabe

Erstelle eine README-Datei, committe sie, ändere sie und committe erneut. Lege zusätzlich eine .gitignore an.

Befehle

```
echo "# Mein Projekt" > README.md
git add README.md
git commit -m "README hinzugefügt"
echo "*.log" > .gitignore
git add .gitignore
git commit -m "gitignore ergänzt"
git log --oneline
```

Ergebnis

git log zeigt mindestens zwei Commits in der Historie.

Technische Hochschule
mannheim

B.2 Änderungen verwalten

Ansehen, vergleichen, rückgängig machen

Änderungen ansehen: status, diff, log

Technische Hochschule mannheim

Aktueller Zustand

```
git status
```

Unterschiede im Detail

```
git diff             # Working Dir vs. Staging
git diff --staged    # Staging vs. letzter Commit
```

Historie

```
git log --oneline --graph --all
git show <commit>    # Details eines Commits
```

Rückgängig machen – der richtige Befehl

Technische Hochschule mannheim

Arbeitskopie verwerfen (uncommitted)

```
git restore <datei>
```

Aus dem Staging nehmen (nicht verwerfen)

```
git restore --staged <datei>
```

Commit rückgängig, Änderungen behalten

```
git reset --soft HEAD~1
```

Öffentlichen Commit sicher zurücknehmen

```
git revert <commit>   # neuer Gegen-Commit
```

Vorsicht

git reset --hard verwirft Änderungen unwiderruflich

Übung · Änderungen rückgängig machen

Technische Hochschule mannheim

Aufgabe

Ändere die README, verwirf die Änderung wieder. Committe dann etwas und nimm den Commit mit revert zurück.

Befehle

```
echo "Tippfehler" >> README.md
git restore README.md
echo "Zeile" >> README.md && git commit -am "Zeile"
git revert HEAD --no-edit
git log --oneline
```

Ergebnis

Die verworfene Änderung ist weg; revert erzeugt einen zusätzlichen Gegen-Commit.

Technische Hochschule
mannheim

B.3 Branching, Merging & Konflikte

Parallel arbeiten ohne Chaos

Zusammenarbeit im Team – das Grundmuster

Technische Hochschule mannheim

Ausgangslage

Ein gemeinsames Remote-Repository (origin), z. B. auf GitHub; alle haben eine lokale Kopie

Grundregel: niemand arbeitet direkt auf main

Jede Aufgabe erhält einen eigenen Branch (Feature-Branch)

Der Ablauf pro Aufgabe

Aktuellen Stand holen (pull) → Branch anlegen → committen → pushen

Pull Request öffnen → Review → in main mergen → Team holt neuen Stand (pull)

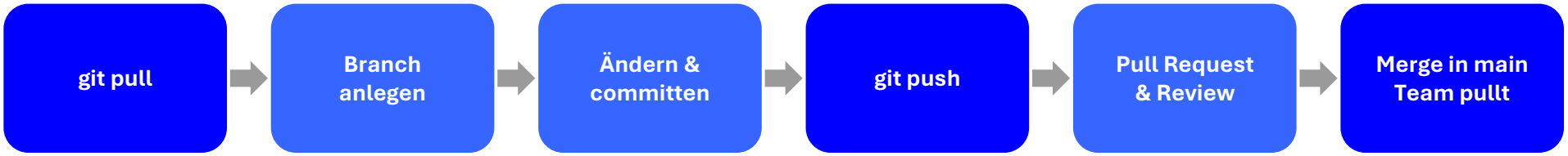
Warum so?

Isolierte Arbeit, nachvollziehbare Reviews, main bleibt stets lauffähig

Der Zusammenarbeits-Zyklus

Technische Hochschule mannheim

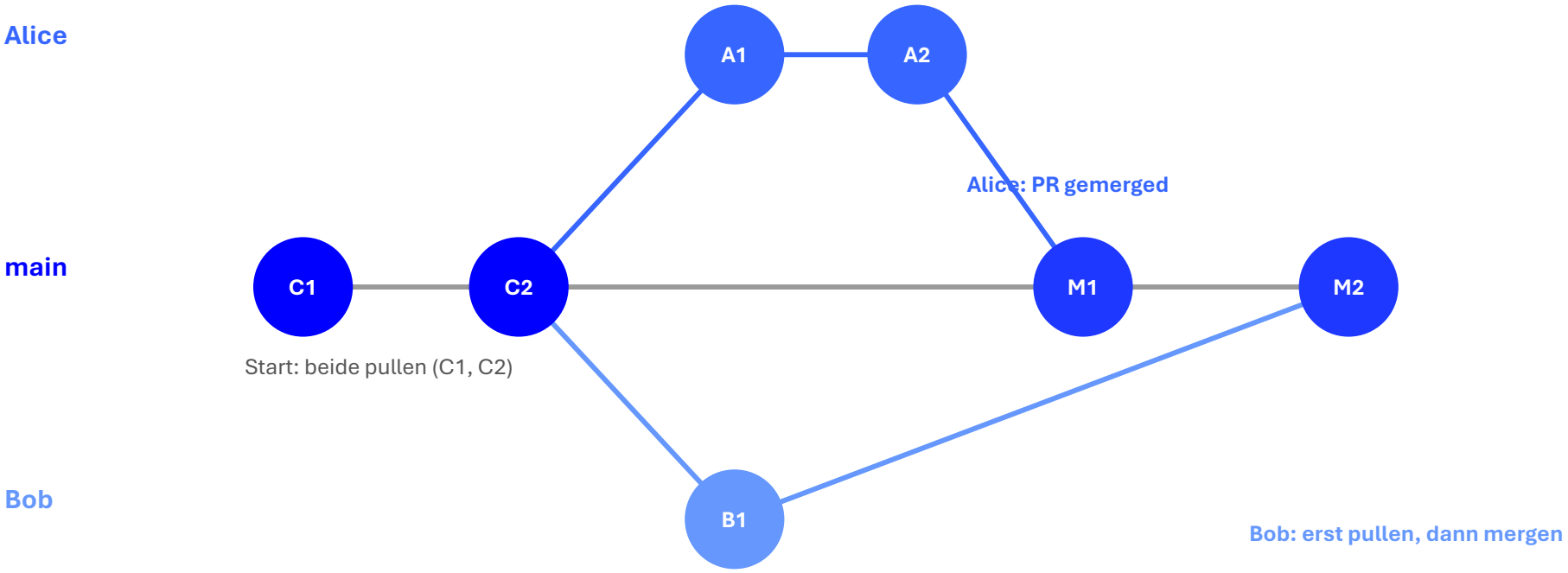
Ein Feature-Branch pro Aufgabe – main bleibt immer lauffähig.



Danach beginnt der Zyklus für die nächste Aufgabe von vorn.

Beispiel: Alice und Bob arbeiten parallel

Technische Hochschule mannheim



A1/A2 und B1 entstehen parallel auf getrennten Branches. Alice merged zuerst (M1); Bob integriert danach den neuen main-Stand und merged (M2).

Beispiel: Schritt für Schritt

Technische Hochschule mannheim

1 · Beide starten auf dem aktuellen Stand

```
git switch main && git pull
```

2 · Alice: Feature-Branch, arbeiten, pushen

```
git switch -c feature-login
git commit -am "Login-Formular" && git push -u origin feature-login
```

3 · Alice öffnet Pull Request → Review → Merge in main

4 · Bob holt den neuen Stand, bevor er merged

```
git switch main && git pull
git switch feature-suche && git merge main   # ggf. Konflikt lösen
git push   # danach PR und Merge
```

Technische Hochschule mannheim

Goldene Regeln der Zusammenarbeit

1

Oft synchronisieren

Vor der Arbeit: git pull

Klein & häufig pushen

Konflikte bleiben klein

2

Sauber trennen

Branch pro Aufgabe

Nie direkt auf main

Pull Request + Review

3

Konflikte früh lösen

main vor Merge holen

Konflikt lokal klären

Erst dann pushen

Branches – parallele Entwicklungslinien

Technische Hochschule mannheim

Idee

Ein Branch ist eine unabhängige Linie von Commits – ideal für Features/Bugfixes

Branch anlegen und wechseln

```
git branch feature-login
git switch feature-login   # (früher: git checkout)
git switch -c feature-login  # anlegen + wechseln
```

Übersicht

```
git branch          # lokale Branches
git log --graph --oneline --all
```

Mergen – Branches zusammenführen

Technische Hochschule mannheim

Vorgehen

Auf den Zielbranch wechseln, dann den Feature-Branch mergen

```
git switch main
git merge feature-login
```

Fast-Forward

Zeiger rückt vor, wenn main keine eigenen neuen Commits hat

Merge-Commit

Bei divergierten Branches entsteht ein zusammenführender Commit

Aufräumen

```
git branch -d feature-login
```

Übung · Feature-Branch & Merge

Technische Hochschule mannheim

Aufgabe

Erstelle einen Branch, mache dort einen Commit und führe ihn in main zusammen.

Befehle

```
git switch -c feature-x
echo "Feature" >> feature.txt
git add feature.txt && git commit -m "Feature X"
git switch main
git merge feature-x
git branch -d feature-x
```

Ergebnis

Der Feature-Commit ist in main sichtbar (git log --oneline).

Merge-Konflikte lösen

Technische Hochschule mannheim

Wann?

Wenn dieselbe Stelle in beiden Branches unterschiedlich geändert wurde

Git markiert die Konfliktstelle

```
<<<<<<< HEAD  … eigene Version
=======
>>>>>>> feature-x  … andere Version
```

Lösen

Datei manuell bereinigen, Marker entfernen

```
git add <datei>   # als gelöst markieren
git commit         # Merge abschließen
```

Übung · Einen Konflikt lösen

Technische Hochschule mannheim

Aufgabe

Ändere in zwei Branches dieselbe Zeile, merge sie und löse den entstehenden Konflikt.

Befehle

```
# in main und in branch-b dieselbe Zeile ändern
git merge branch-b     # Konflikt!
git status             # zeigt Konfliktdatei
# Datei editieren, Marker entfernen
git add <datei>
git commit
```

Ergebnis

Der Merge ist abgeschlossen; die Datei enthält die gewünschte, bereinigte Fassung.

Technische Hochschule mannheim

Merge vs. Rebase (Überblick)

1

Merge

Führt Zweige zusammen

Erhält die Historie

Merge-Commit möglich

Sicher & einfach

2

Rebase

Setzt Commits neu auf

Lineare Historie

git rebase main

„Sauberer“ Verlauf

3

Goldene Regel

Bereits geteilte

(gepushte) Historie

niemals rebasen

Technische Hochschule
mannheim

B.4 Remotes & Zusammenarbeit

Im Team arbeiten – mit GitHub & Co.

Remotes – geteilte Repositories

Technische Hochschule mannheim

Remote

Ein Repository auf einem Server; die Standardbezeichnung ist „origin“

Verbinden / ansehen

```
git remote add origin <url>
git remote -v
```

Austausch

```
git push -u origin main   # hochladen (erstmalig)
git fetch                 # holen ohne mergen
git pull                  # holen + integrieren
```

Wie funktioniert GitHub?

Technische Hochschule mannheim

Was ist GitHub?

Web-Plattform zum Hosten von Git-Repositories inkl. Zusammenarbeit, Issues, Reviews und CI/CD (GitLab/Bitbucket funktionieren ähnlich)

Grundbausteine

Konto → Repository (öffentlich/privat) → Klonen/Pushen

Remote lokal ↔ GitHub

Lokales Repo per push hochladen, Änderungen anderer per pull holen

Sozialer Aspekt

Fork (eigene Kopie), Stars, Issues, Discussions

GitHub: Authentifizierung & erster Push

Technische Hochschule mannheim

HTTPS mit Personal Access Token (PAT)

Passwort-Login ist deaktiviert – Token statt Passwort verwenden

oder SSH-Schlüssel

```
ssh-keygen -t ed25519 -C "du@example.com"
# öffentlichen Schlüssel in GitHub → Settings → SSH Keys
```

Repo verbinden und pushen

```
git remote add origin git@github.com:user/repo.git
git push -u origin main
```

Pull Requests & Code Review

Technische Hochschule mannheim

Pull Request (PR) / Merge Request

Antrag, einen Branch in den Zielzweig zu übernehmen

Typischer Ablauf

Branch pushen → PR öffnen → Review → Diskussion/Änderungen → Merge

Review

Andere prüfen die Änderungen: Qualität, Stil, Fehler, Wissenstransfer

Automatische Checks

CI-Pipeline (Build, Tests, Linter) muss grün sein

Übung · Zu GitHub pushen & PR öffnen

Technische Hochschule mannheim

Aufgabe

Lege auf GitHub ein leeres Repo an, verbinde dein lokales Repo und pushe. Öffne dann für einen Branch einen Pull Request.

Befehle

```
git remote add origin <url>
git push -u origin main
git switch -c feature-doku
git commit -am "Doku ergänzt"
git push -u origin feature-doku
# auf GitHub: „Compare & pull request“
```

Ergebnis

Der Branch ist auf GitHub sichtbar und ein Pull Request ist geöffnet.

Technische Hochschule
mannheim

B.5 Workflows & Best Practices

Nachhaltig und im Team arbeiten

Technische Hochschule
mannheim

Verbreitete Git-Workflows

1

GitHub Flow

main + Feature-Branches

Pull Requests

Einfach

Für Continuous Delivery

2

Git Flow

main + develop

feature/release/hotfix

Strukturiert

Für Releasezyklen

3

Trunk-Based

Ein Hauptzweig

Sehr kurze Branches

Feature Flags

Stark CI-getrieben

Gute Commits schreiben

Technische Hochschule mannheim

Kleine, fokussierte Commits

Eine logische Änderung pro Commit – leichter zu prüfen und zurückzurollen

Aussagekräftige Nachrichten

Kurze Betreffzeile (imperativ), bei Bedarf Absatz mit dem „Warum“

`Beispiel: "Fix: Login schlägt bei leerem Passwort fehl"`

Häufig committen

Kleine Schritte statt riesiger Sammel-Commits

Nicht committen

Secrets, Passwörter, generierte Artefakte

Best Practices & häufige Fehler

Technische Hochschule mannheim

Häufig pushen und pullen

Integration nicht aufschieben – Konflikte klein halten

Branch pro Aufgabe

Feature/Bugfix isoliert entwickeln, dann per PR mergen

Vor der Arbeit aktualisieren

git pull, um auf dem aktuellen Stand zu starten

Typische Fehler

Große Binärdateien einchecken; .env/Secrets committen; auf main direkt arbeiten

Spickzettel – die wichtigsten Befehle

Technische Hochschule mannheim

```
git init / git clone <url>          # Repo anlegen/holen
git status / git diff               # Zustand & Änderungen
git add <datei> / git add .         # stagen
git commit -m "…"                   # committen
git switch -c <branch>              # Branch anlegen+wechseln
git merge <branch>                  # zusammenführen
git pull / git push                 # mit Remote abgleichen
git log --oneline --graph --all     # Historie ansehen
```

Technische Hochschule
mannheim

Zusammenfassung & Ressourcen

Zusammenfassung

Technische Hochschule mannheim

Versionskontrolle gibt dir Historie, Sicherheit und Teamfähigkeit

Git ist verteilt: jede Kopie hat die volle Historie

Merke den Datenfluss: Working Directory → Staging → Repository

Der Alltag: edit → add → commit – und regelmäßig pushen/pullen

Branches ermöglichen paralleles, gefahrloses Arbeiten

Remotes & Pull Requests strukturieren die Zusammenarbeit

Kleine Commits, klare Nachrichten, sauberer Workflow

Ressourcen zum Weiterlernen

Technische Hochschule mannheim

Pro Git (Chacon/Straub) – kostenloses Buch: git-scm.com/book

Offizielle Doku & Referenz: git-scm.com/doc

Interaktiv üben: learngitbranching.js.org

GitHub Docs & Skills: docs.github.com

Spickzettel: education.github.com/git-cheat-sheet

Technische Hochschule
mannheim

Viel Erfolg mit Git!

Fragen und gemeinsames Üben jederzeit willkommen