## Fragen an ein Versionskontrollsystem (VCS)

- Wie sah mein Code vor einer Woche / bei Release 1.0 aus?
- Seit welchem Release ist dieses Feature / dieser Bug drin?
- Warum wurde dieser Code-Teil eingefügt?
- Wen kann ich zu diesem Code-Teil befragen?
- Wie kann ich Änderungen mehrerer Mitarbeiter zusammenführen?

## Welche Informationen brauchen wir?

- Wer hat
- Wann
- Was (Basisversion)
- Wie (neue Version)
- Warum geändert?

## Verteilte Versionskontrolle (DVCS)

- jeder Anwender hat eine Kopie der Historie
- viele Operationen sind dadurch deutlich schneller
- Offline-Arbeit möglich
- Änderungen "schnell mal eben" durchführen und später aufräumen
- Beispiele: Git, Mercurial, Bazaar, Darcs, …

## Wir begeben uns an den Start

```
git config --global user.name "Dein Name"
git config --global user.email user@domain.tld
```

(wichtig für die *Wer* Frage)

```
git help config
```

git help `Kommando` zeigt die Hilfe zu einem beliebigen Git-Kommando an (wichtig wenn man kein photographisches Gedächtnis hat).

## Auf die Plätze …

```
mkdir repo
cd repo
git init
#Initialisierte leeres Git-Repository in /home/knut/repo/.git/
```

```
echo Tübix > cool_stuff.txt
git status
#Auf Branch master
#
#Initialer Commit
#
#Unversionierte Dateien:
#  (benutzen Sie "git add <Datei>...", um die Änderungen zum
#Commit vorzumerken)
#
#   cool_stuff.txt
#
#nichts zum Commit vorgemerkt, aber es gibt unversionierte
#Dateien (benutzen Sie "git add" zum Versionieren)
```

### ... fertig ...

```
git add cool_stuff.txt
git status
#Auf Branch master
#
#Initialer Commit
#
#zum Commit vorgemerkte Änderungen:
#  (benutzen Sie "git rm --cached <Datei>..." zum Entfernen
#aus der Staging-Area)
#
#   neue Datei:     cool_stuff.txt
#
```

### ... los!

```
git commit
#[master (Basis-Commit) b323f69] Mein erster Git-Commit
# 1 file changed, 1 insertion(+)
# create mode 100644 cool_stuff.txt

git log -p
#commit b323f69414bcc1d448c475bf07a7e6a439414da2
#Author: Knut Franke <Knut.Franke@gmx.de>
#Date:   Fri Jun 10 19:20:58 2016 +0200
#
#    Mein erster Git-Commit
#
```

```
#diff --git a/cool_stuff.txt b/cool_stuff.txt
#new file mode 100644
#index 0000000..545e14c
#--- /dev/null
#+++ b/cool_stuff.txt
#@@ -0,0 +1 @@
#+Tübix
```

Klasse, wir können Git!

## Haben wir alles was wir brauchen?

```
git log --graph
#* commit 3adcc804dce0690303236b53971d3fd0701537dc
#| Author: Knut Franke <Knut.Franke@gmx.de>
#| Date:   Fri Jun 10 19:26:09 2016 +0200
#|
#|     Noch mehr tolle Sachen
#|
#* commit b323f69414bcc1d448c475bf07a7e6a439414da2
#  Author: Knut Franke <Knut.Franke@gmx.de>
#  Date:   Fri Jun 10 19:20:58 2016 +0200
#
#      Mein erster Git-Commit
```

- *Wer*, *Wann*, *Was*, *Wie* geht automatisch
- Was ist mit *Warum*?

## Exkurs: Commit-Messages

- sollen i.d.R. die *Warum* Frage beantworten (falls nicht trivial)
- können zu diesem Zweck Verweise auf Ticket-ID o.ä. enthalten
- sollen Änderungen für andere nachvollziehbar machen - und für dich nach dem nächsten Urlaub!

Konvention zur Formatierung:

```
Kurze Zusammenfassung (ca. 50 Zeichen, Was-Frage)

Leerzeile, dann eine (ggf. längere) Erklärung, die die
Warum-Frage beantwortet.

Bei Änderungen mit nicht-offensichtlichen Stolperfallen eventuell
auch noch erklären, welche Alternativen warum verworfen wurden.
```

## Memo

| Befehl | Zweck |
|---|---|
| git help `cmd` | zeigt Dokumentation zu `cmd` |
| git init | legt neues Repository an |
| git status | zeigt Zustand des Repos an, gibt Tipps |
| git diff | zeigt noch nicht vorgemerkte Änderungen |
| git add … | macht Snapshot von Datei(en) für nächsten Commit |
| git commit | speichert vorgemerkte Änderungen |
| git log | zeigt Commit-Historie |

## Run 1: Basis-Workflow

```
git config --global user.name "Dein Name"
git config --global user.email user@domain.tld
```

1. lege ein neues Git-Repository an
2. committe ein paar Änderungen
3. schau dir die Commit-Historie an
4. Kür: schau dir bei jedem Schritt (z.B. mit `find`) an, was sich im `.git` Verzeichnis ändert

## Zentrales Repo

```
git clone http://lab/tuebix/hallo.git
#Klone nach 'hallo' ...
#remote: Counting objects: 6, done.
#remote: Compressing objects: 100% (2/2), done.
#remote: Total 6 (delta 0), reused 0 (delta 0)
#Entpacke Objekte: 100% (6/6), Fertig.
#Prüfe Konnektivität ... Fertig.
cd hallo
ls
#cool_stuff.txt

git add zahlen.txt
git commit
#[master e8fa16e] Zahlen-Sammlung gestartet
# 1 file changed, 1 insertion(+)
# create mode 100644 zahlen.txt
```

## Updates holen

```
git pull
#Merge made by the 'recursive' strategy.
# cool_stuff.txt | 1 +
# 1 file changed, 1 insertion(+)
```

Was genau ist hier passiert?

```
git log --graph --oneline
#*   8c577b0 Merge branch 'master' of http://lab/tuebix/hallo
#|\
#| * b983d4c Linux ist auch cool
#* | e8fa16e Zahlen-Sammlung gestartet
#|/
#* 3adcc80 Noch mehr tolle Sachen
#* b323f69 Mein erster Git-Commit
```

## Konflikte

```
git pull
#automatischer Merge von cool_stuff.txt
#KONFLIKT (Inhalt): Merge-Konflikt in cool_stuff.txt
#Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte
#und committen Sie dann das Ergebnis.
```

In `cool_stuff.txt`:

```
Tübix
Git
<<<<<<< HEAD
reveal.js
=======
Linux
>>>>>>> b983d4cc25261780d3b2a21a142d4750394c5c77
```

## Manuelle Konfliktlösung

1. Datei im Editor öffnen und alle Konflikstellen bearbeiten:

`conflict` `Tübix` `Git` `reveal.js` `Linux` 2. `git add cool_stuff.txt`
3. `git commit`

### Alternativ: git-mergetool

```
git mergetool
#
#This message is displayed because 'merge.tool' is not configured.
#See 'git mergetool --tool-help' or 'git help config' for more
#details. 'git mergetool' will now attempt to use one of the
#following tools:
#opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff
#diffuse diffmerge ecmerge p4merge araxis bc codecompare vimdiff
#emerge
#Merging:
#cool_stuff.txt
#
#Normal merge conflict for 'cool_stuff.txt':
#  {local}: modified file
#  {remote}: modified file
#Hit return to start merge resolution tool (gvimdiff):
```

```
git commit
#[master ad06154] Merge branch 'master' of http://lab/tuebix/hallo
```

### Änderungen “hochladen”

```
git push
#Username for 'http://lab': anon
#Password for 'http://anon@lab':
#Zähle Objekte: 6, Fertig.
#Delta compression using up to 8 threads.
#Komprimiere Objekte: 100% (2/2), Fertig.
#Schreibe Objekte: 100% (6/6), 588 bytes | 0 bytes/s, Fertig.
#Total 6 (delta 0), reused 0 (delta 0)
#To http://lab/tuebix/hallo.git
#   b983d4c..ad06154  master -> master
```

```
git push
#Username for 'http://lab': anon
#Password for 'http://anon@lab':
#To http://lab/tuebix/hallo.git
# ! [rejected]        master -> master (non-fast-forward)
#error: Fehler beim Versenden einiger Referenzen nach 'http://lab/tuebix/hallo.git'
#Hinweis: Aktualisierungen wurden zurückgewiesen, weil die Spitze Ihres aktuellen
#Hinweis: Branches hinter seinem externen Gegenstück zurückgefallen ist. Führen Sie
#Hinweis: die externen Änderungen zusammen (z.B. 'git pull ...') bevor Sie "push"
```

```
#Hinweis: erneut ausführen.
#Hinweis: Siehe auch die Sektion 'Note about fast-forwards' in 'git push --help'
#Hinweis: für weitere Details.
```

## Memo

| Befehl | Zweck |
|---|---|
| git clone … | Legt eine Kopie des angegebenen Repos (“origin”) an |
| git pull | holt Aktualisierungen aus “origin” |
| git push | kopiert eigene Änderungen nach “origin” |
| git mergetool | automatisiert die Auflösung von Konflikten |

## Run 2: Zentrales Repository

1. hole dir eine Kopie von http://lab/tuebix/hallo.git
2. füge eine Datei hinzu und committe (bitte die vorhandene Datei jetzt noch nicht ändern!)
3. Lade deine Änderungen hoch und hole dir die Änderungen der anderen Teilnehmer (User: `anon` Passwort: `tuebix16`)
4. das gleiche nochmal mit der vorhandenen Datei `cool_stuff.txt` (das sollte Merge-Konflikte geben, die du auflösen musst, bevor du pushen kannst - falls nicht, versuch es nochmal mit einer weiteren Änderung, nachdem jemand anderes gepusht hat)

## Exkurs: Unsere GitLab-Spielwiese

1. lege dir unter http://lab/users/sign_in einen User an
2. die Aufforderung zur “confirmation email” ignorieren und direkt unter http://lab/users/sign_in einloggen
3. optional: SSH-Key hinterlegen

## Run 3: Merge Requests

1. öffne http://lab/tuebix/hallo im Browser und klicke auf “Fork”, dann auf deinen Namen
2. den neuen Fork klonen (http://lab/deinuser/hallo.git, oder git@lab:deinuser/hallo.git falls du einen Key hinterlegt hast)
3. führe eine Änderung durch und pushe sie in deinen Fork
4. rechts neben der URL von deinem Fork auf das + klicken und “New merge request” wählen

5. links unter “select source branch” den Eintrag “master” wählen, weiter durchklicken

## Auslauf: Verwendete Tools

- Projekt-Server/Spielwiese: GitLab CE auf Raspberry PI 3
- Markup-Sprache: Markdown
- Präsentation: reveal.js
- PDF Export: Pandoc