Basics of version control system git

1st week	2nd week
Intro and Overview	Versioning (Branches)
View, Navigate, Create, Change	Publishing
Installation	Collaboration
Versioning (Basics)	Review

Versionierung

Beim Entwickeln von Software **braucht** man ein Versioning System

- Viele Leute aus einem großen Team arbeiten in einem Projekt zusammen
- Alte Versionen aufrecht erhalten, neue Versionen entwickeln
- Problemlösung in unterschiedlichen Versionen
- Meist gibt es parallel mehrere Versionen vor Release:
  - Entwicklung
  - Testing
  - Marketing

Versioning

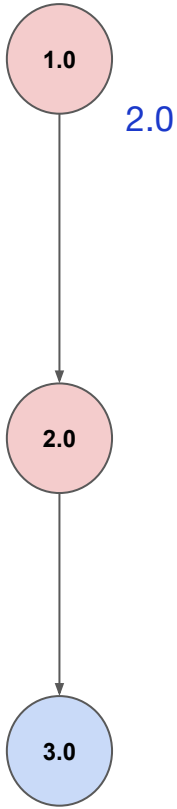
You are developing a mobile game “**Cyborg**” 🤖.

You release 1.0, and then 2.0

Versioning

You are developing a mobile game “**Cyborg**” 🤖.

You release 1.0, and then 2.0
You start 3.0 and find a small **bug** in 1.0 & 2.0

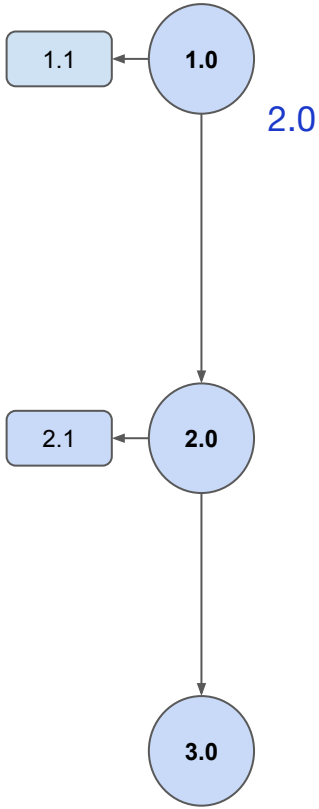


DCI Versioning

You are developing a mobile game “**Cyborg**” 🤖.

You release 1.0, and then 2.0
You start 3.0 and find a small **bug** in 1.0 & 2.0

Some still use 1.0 (old phones can’t use 2.0) so fixes are made → 1.1 & 2.1 & 3.0



Versioning

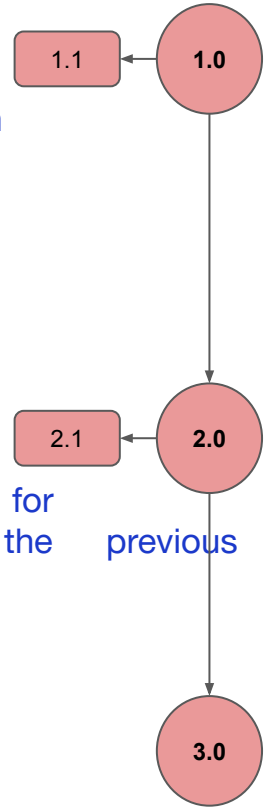
You are developing a mobile game “**Cyborg**” 🤖.

You release 1.0, and then 2.0
You start 3.0 and find a small **bug** in 1.0 & 2.0

Some still use 1.0 (old phones can’t use 2.0) so fixes are made → 1.1 & 2.1 & 3.0

You find a **critical bug** affecting 1.0 / 1.1 / 2.0 / 2.1 / 3.0

The fix is the same for all
you need an emergency fix for **all** versions, even ones without the previous fix



Versioning

You are developing a mobile game “**Cyborg**” 🤖.

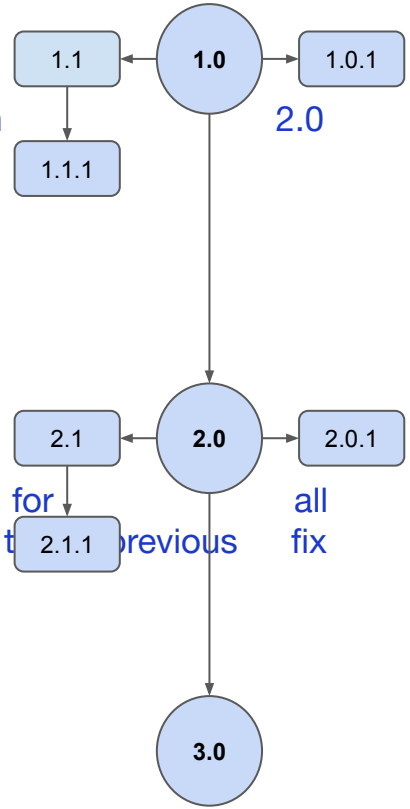
You release 1.0, and then 2.0
You start 3.0 and find a small **bug** in 1.0 & 2.0

Some still use 1.0 (old phones can’t use 2.0) so fixes are made → 1.1 & 2.1 & 3.0

You find a **critical bug** affecting 1.0 / 1.1 / 2.0 / 2.1 / 3.0

The fix is the same for all
you need an emergency fix for **all** versions, even ones without the previous fix

Now you have 1.0.1, 1.1.1, 2.0.1, 2.1.1 and 3.0. 😨

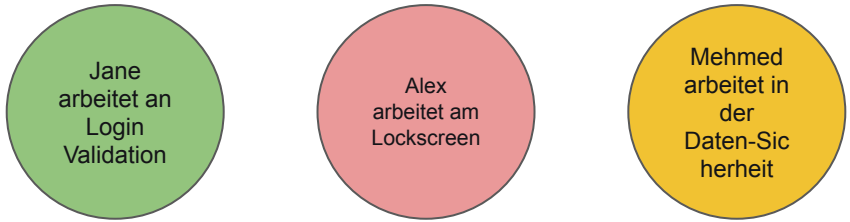


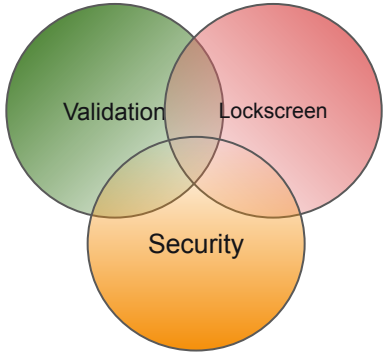
Ein System wird gebraucht!

Versioning

Ein anderes Szenario für **Cyborg**!

Was passiert bei mehreren Entwicklern?

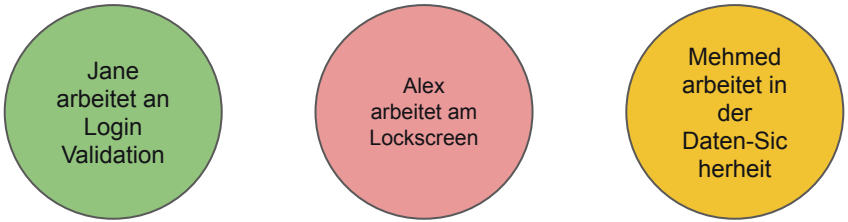


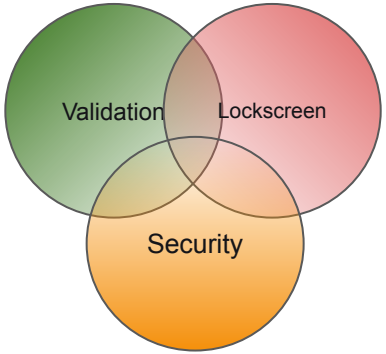


Versioning

Ein anderes Szenario für **Cyborg**!

Was passiert bei mehreren Entwicklern?





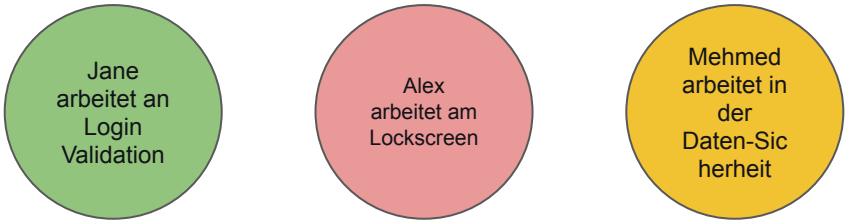
Drei Versionen mit überlappendem Inhalt!

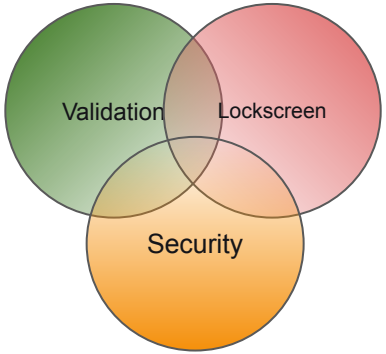
Mehmed beendet seine Arbeit → Jane und Alex müssen ihre Versionen updaten
Jane und Alex mergen die Arbeit von Mehmed → Konflikt-Code!

Versioning

Ein anderes Szenario für **Cyborg**!

Was passiert bei mehreren Entwicklern?





Drei Versionen mit überlappendem Inhalt!

Mehmed beendet seine Arbeit → Jane und Alex müssen ihre Versionen updaten
Jane und Alex mergen die Arbeit von Mehmed → Konflikt-Code!

Jetzt stell dir das mit 18 Entwicklern vor!

Ein System wird gebraucht!

Versioning

Das Problem wird gelöst durch ein **Version Control System** (VCS)

- Versionskontrolle
- Revisionskontrolle
- Source Control
- Source Code Management

Versioning

Das Problem wird gelöst durch ein **Version Control System** (VCS)

- Versionskontrolle
- Revisionskontrolle
- Source Control
- Source Code Management

Hauptsächlich für Source Code genutzt, aber kann auch für jedes andere Versioning genutzt werden

- Dokumente wie Markdown Files
- Dokumentationen
- Data Files - wie Übersetzungen
- Konfigurationsdateien

Versioning

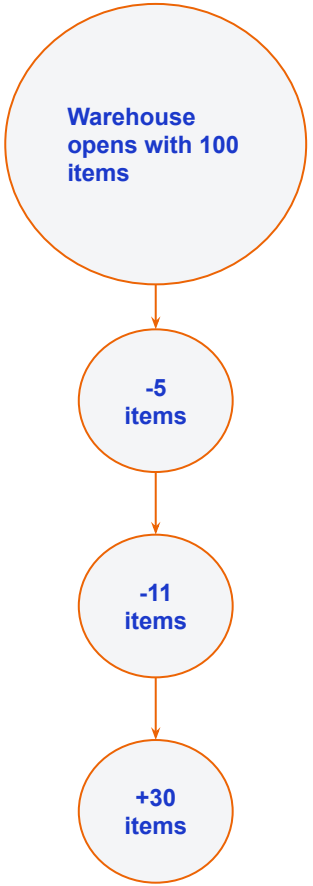
Ein **Version Control System** ist wie ein Warenhaus Log Keeper

“**Antero**” arbeitet in einem Warenhaus

Antero führt einen Inventar Log

Wenn Kunden etwas kaufen, loggt Anton die Veränderung!

Wenn eine Lieferung eintrifft, loggt Anton die Veränderung!



Versioning

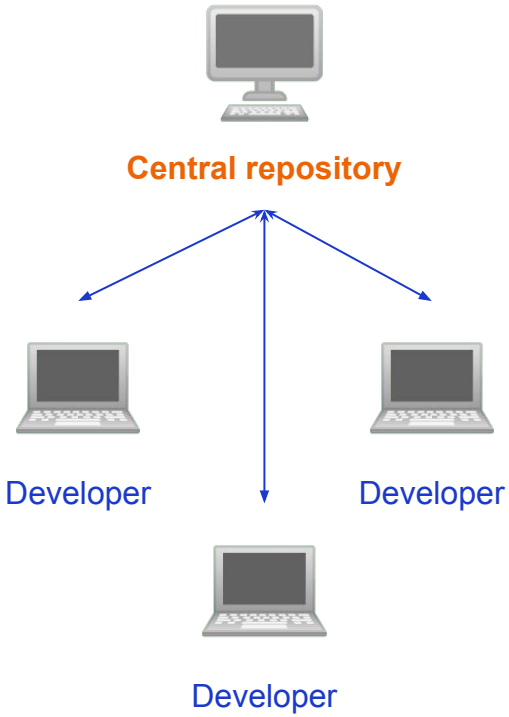
Früher nutzte man als Version Control System CVS *(1986)*

CVS hatte ein **client-server Modell**

Ein Server speichert Dateien und Datei-Historie in einem zentralen Repository (Repo)

Entwickler checken Kopien aus dem zentralen Repo aus

Nach der Arbeit checken die Entwickler ihre Veränderungen ein



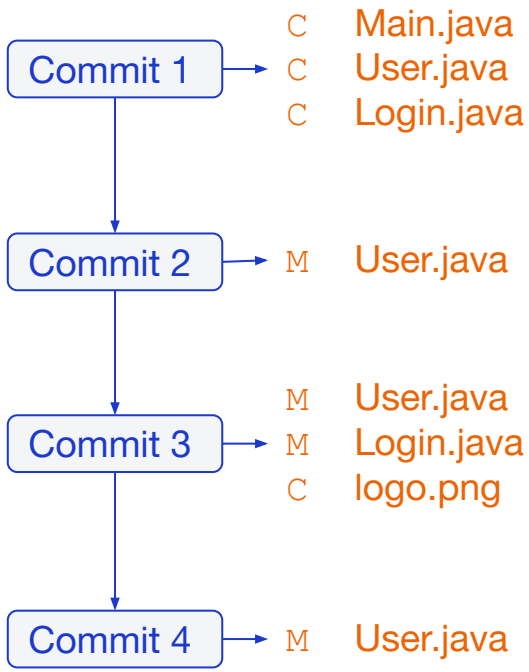
Versioning

CVS trackt Veränderungen via Datei
→ kannte also nur die Veränderungen einzelner Dateien

Die nächste Generation: **Subversion** (svn)

SVN hat ganze Sets an Veränderungen getrackt,
sogenannte **Commits**

Man trackt so also die Historie des gesamten Projekts
Veränderung für Veränderung

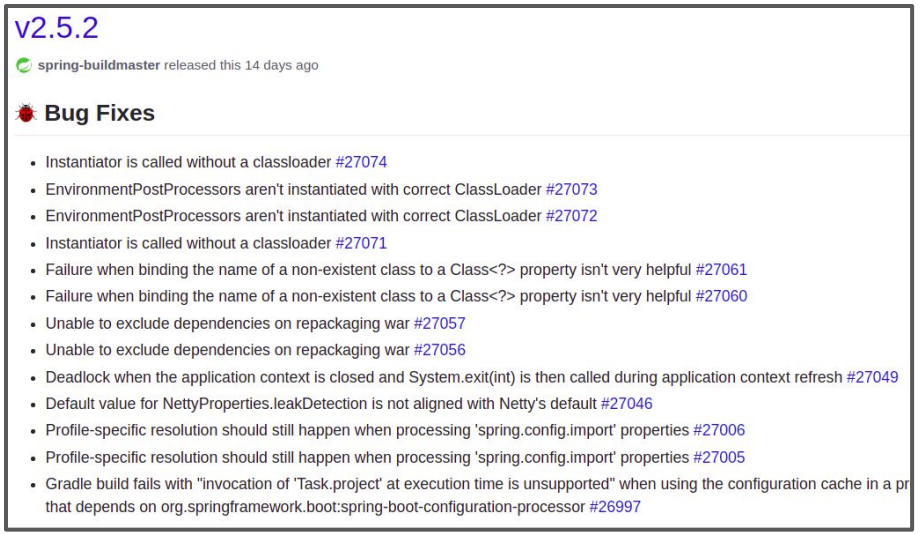


Versioning

Für jeden Commit fügst du eine **commit message** hinzu

Es entsteht mit der Zeit eine Liste aller Veränderungen

Diese Liste an Veränderungen wird *Changelog* genannt



v2.5.2

spring-buildmaster released this 14 days ago

Bug Fixes

- Instantiator is called without a classloader #27074
- EnvironmentPostProcessors aren't instantiated with correct ClassLoader #27073
- EnvironmentPostProcessors aren't instantiated with correct ClassLoader #27072
- Instantiator is called without a classloader #27071
- Failure when binding the name of a non-existent class to a Class<?> property isn't very helpful #27061
- Failure when binding the name of a non-existent class to a Class<?> property isn't very helpful #27060
- Unable to exclude dependencies on repackaging war #27057
- Unable to exclude dependencies on repackaging war #27056
- Deadlock when the application context is closed and System.exit(int) is then called during application context refresh #27049
- Default value for NettyProperties.leakDetection is not aligned with Netty's default #27046
- Profile-specific resolution should still happen when processing 'spring.config.import' properties #27006
- Profile-specific resolution should still happen when processing 'spring.config.import' properties #27005
- Gradle build fails with "invocation of 'Task.project' at execution time is unsupported" when using the configuration cache in a pr
  that depends on org.springframework.boot:spring-boot-configuration-processor #26997

Viele Unternehmen haben Richtlinien, wie die Commit Messages aussehen sollen

At the core of the lesson

- Version Control Systems helfen mit
  - Kollaborationen
  - Release Management

- Meist gibt es ein zentrales Repository

- Moderne Systeme tracken Commits
- Ein Commit ist ein Set aus Dateiänderungen
- Commits haben eine Commit Message

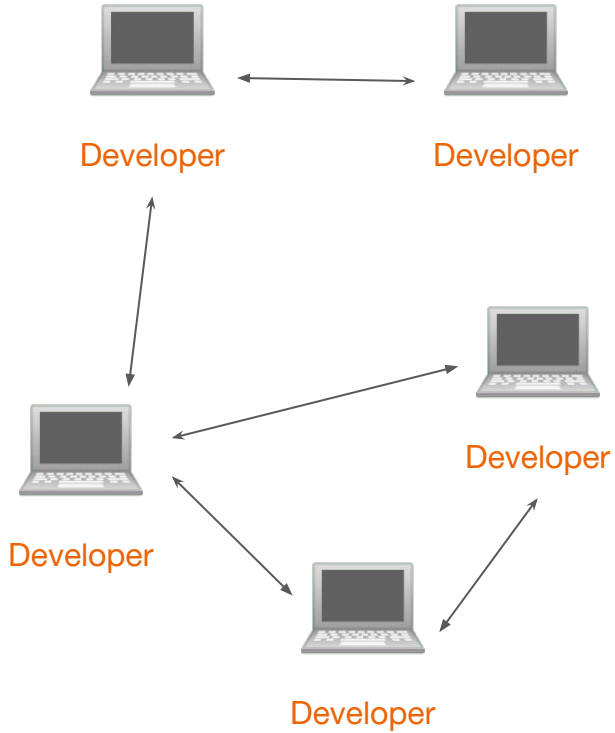
GIT

Versioning

Subversion war die Generation vor VCS

Und **git** ist die aktuelle Generation eines **Versionsverwaltungssystems**:

- Viele Konzepte sind gleichbleibend, wie Commits
- Git ist ***distributed***/ *verteilt* (DVCS)*
  - Ursprünglich gab es ein zentrales Repo
  - In git hast du *nur* Repos
  - Du checkst ein Repo nicht aus, sondern du klonst es



Versioning

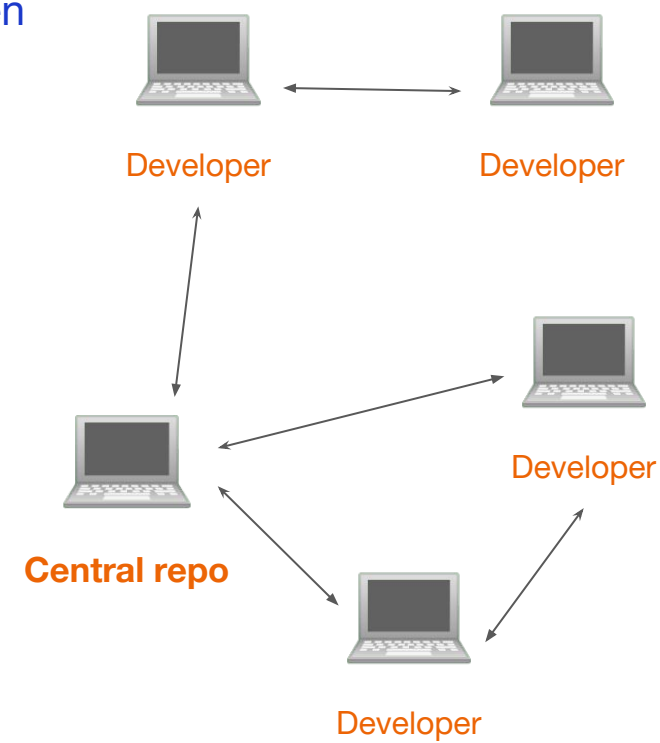
Du kannst Arbeitsstände zwischen Entwicklern transferieren

Normalerweise gibt es weiterhin ein zentrales Repo

Zentrale Repositories können intern oder bei einem externen Service Provider liegen

Es gibt einige Git Services:

- GitHub
- GitLab
- Bitbucket
- SourceForge



Versioning

- Used in practically all modern software development
- Decentralized / Distributed
- Free and open source
- Scales well for large projects
  - Linux 
  - Visual Studio Code
  - React
- Many tools and services
  - Hosting services
  - Visualization tools
  - Integrations
- Rapid branching *(we will discuss branching soon)*

Versioning

Der reguläre Weg Git zu nutzen ist das Git CLI

- Command Line Interface
- Wird nur im Terminal genutzt (durch das **git** Programm)

Wichtige Konzepte:

a. Jeder Ordner kann zu einem neuen git Repository werden
b. Oder Repositories können von anderen Repos geklont werden
c. Achte darauf, Repos **nicht** in Repos anzulegen (nesting)
d. Nutze nicht `sudo` mit git

Achte darauf, in welchem Ordner du arbeitest!
Nutze einen Pfad für all deine Git-Repos: ~/projects/

Versioning

Git muss noch konfiguriert werden

Git Konfigurationen sind gespeichert in **~/.gitconfig**

Du kannst die Datei bearbeiten oder das Command nutzen: **$ git config**

```
git config --global user.name "Joel Peltonen"
git config --global user.email "joel.peltonen@example.com"
```

Note: Es gibt viele weitere config options!

Versioning

Du kannst deinen aktuellen Arbeitsordner zu einem Git Repository machen:

```
$ git init
```

Dadurch wird ein neues Verzeichnis generiert: ***.git***

```
$ ls -a
```

Um ein Repository zu löschen, lösche das .git Verzeichnis

```
$ rm -rf .git
```

Lasse *git init* nicht in einem alten Repo laufen.
Falls das doch mal passieren sollte, lösche das .git Verzeichnis.
Achte darauf, das richtige Verzeichnis zu löschen!

Versioning

.git beinhaltet:

- Alle Commits in allen Branches
- Die Verknüpfungen/ Beziehungen zu anderen Repositories

Versioning

Das .git Verzeichnis kann nur im Top Level (Root) Ordner des Repositories gefunden werden

Das Verzeichnis für ein Projekt könnte so aussehen:

```
projects/
    my-application/
        .git/                    #Beachte: nur ein .git-Verzeichnis
        src/
        public/
        images/
        docs/
```

At the core of the lesson

- Git wird mit dem **git** Command verwendet
- Git ist “distributed”
  - Normalerweise gibt es ein zentrales Repository
- Jeder Ordner kann ein Repository sein
- **git init** generiert ein Repository
- Repositories werden Repo genannt
- Konfiguriere git mit Email und Name

Basic git commands & workflow

Git workflow

Note: git interessiert sich nur für Dateien

- Erstelle eine Datei → git interessiert sich dafür
- Erstelle einen Ordner → git interessiert das nicht
- Erstelle eine Datei in einem Ordner → ***wow*** git ist interessiert!

Dein Weg, git zu nutzen, wird Workflow genannt

- Workflows können von Unternehmen zu Unternehmen unterschiedlich sein
- Workflows können von Projekt zu Projekt unterschiedlich sein

Git workflow

Der wichtigste git Command:

```
git status
```

Dieser Command verrät dir:

- Ob du in einem git Repository bist
- Den aktuellen git Status
  - Gibt es Veränderungen in Dateien?
  - In welchem Branch befindest du dich?

Git workflow

```
dci@dci-laptop:~/projects/not-awesome-project $ git status
fatal: not a git repository (or any of the parent directories): .git
```

Git workflow

```
dci@dci-laptop:~/projects/not-awesome-project $ git status
fatal: not a git repository (or any of the parent directories): .git
```

```
dci@dci-laptop:~/projects/my-awesome-project $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Git workflow

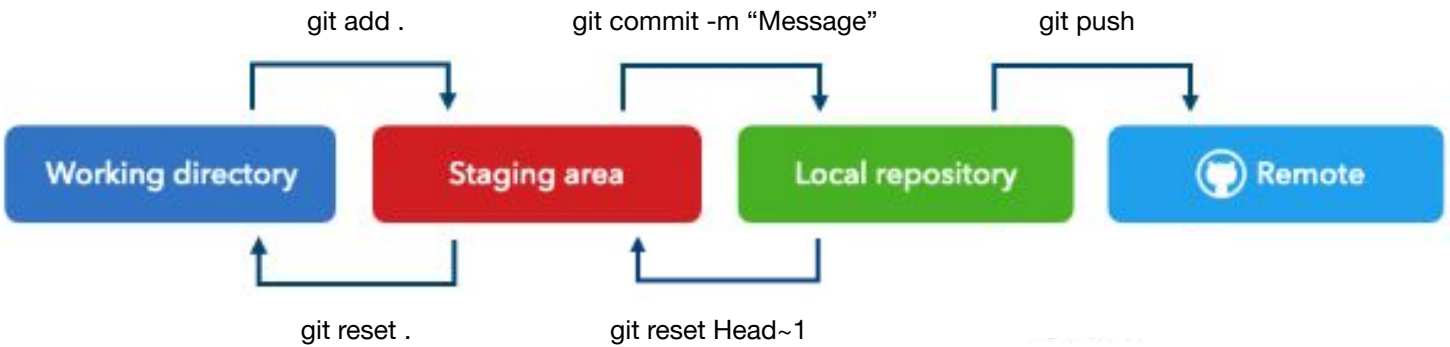
```
dci@dci-laptop:~/projects/my-awesome-project $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.adoc

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    newfile

no changes added to commit (use "git add" and/or "git commit -a")
```

Git workflow



Git workflow

Wenn du bereit für einen Commit bist:

- Musst du git wissen lassen, welche Veränderungen Teil deines Commits sein sollen
- Das machst du mit “**staging**” dieser Veränderungen

```
$ git add <path>
```

Git workflow

```
dci@dci-laptop:~/projects/my-awesome-project$ git add README.adoc
dci@dci-laptop:~/projects/my-awesome-project$ git add newfile
dci@dci-laptop:~/projects/my-awesome-project$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.adoc
        new file:   newfile
```

Git workflow

The *git add* command accepts any path - including folders

What did the . and .. shortcuts mean?

```
dci@dci-laptop:~/projects/my-awesome-project$ git add .
dci@dci-laptop:~/projects/my-awesome-project$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.adoc
    new file:   newfile
```

Git workflow

Denk daran, Commits benötigen eine Commit Message

```
$ git commit
```

Du ergänzt diese Message mit **-m**

```
dci@dci-laptop:~/projects/test$ git commit -m “Add demo feature”

[main 7e1cf288f0] Add demo feature
 2 files changed, 1 deletion(-)
 create mode 100644 newfile
```

Git workflow

Ein Commit ist ein Schnappschuss (Momentaufnahme) deiner Dateien in deinem Repository zu einer bestimmten Zeit.
Dein Commit existiert jetzt **nur** in deinem lokalen Repository

Manche Projekte oder Unternehmen haben Commit Regeln:

- Mache einen Commit nur bei funktionierendem Code
- Mache einen Commit nur bei einem logischen Set an Veränderungen
- max 50 Character
- Nutze ganze Sätze
- Starte deine Message mit einer ID
- Schreib im Präsens: **Fix bug** nicht **Fixed bug**

AD-192 Add autosuggestions to search

Git workflow

Wenn dein Repository geklont oder mit einem anderen Repository verbunden ist, kannst du deinen Commit zum Remote Repository schicken:

$ git push

Git workflow

Um dein lokales Repository zu aktualisieren:

$ git pull

Es kann zu einem Konflikt kommen, wenn du eine Datei geändert hast, die im Remote Repository bereits geändert wurde.

Git workflow

Lese die Commit Historie mit **git log**

Jeder Commit hat eine ID, einen Autor, ein Datum und eine Message
Um den Log zu verlassen, drücke **q**

```
dci@dci-laptop:~/projects/my-awesome-project$ git log
commit 7e1cf288f03e8481b77d9505d9098a994b2bce9e (HEAD -> main)
Author: Joel Peltonen <joel.peltonen@digitalcareerinstitute.org>
Date:   Fri Jul 9 17:45:11 2021 +0200

    Add demo feature

commit 7a1c923fecacd4abafda82fa8c2fc6be3bc4e761 (origin/main, origin/HEAD)
Merge: 0b604f5e3b 3de58c2340
Author: Andy Example <example@example.org>
Date:   Fri Jul 9 14:18:18 2021 +0100

    Merge branch '2.5.x'

     Closes gh-27226
```

At the core of the lesson

```
$ git status      # what is happening
$ git add         # stage changes
$ git commit      # commit changes
$ git push        # push to remote
$ git pull        # pull from remote
$ git log         # view history
```

Branching

1st week	2nd week
Intro and Overview	Versioning (Branches)
View, Navigate, Create, Change	Publishing
Installation	Collaboration
Versioning (Basics)	Review