Dezentrale Versionsverwaltung

mit GIT

Versionsverwaltung?

Versionsverwaltung?

- Speichern unterschiedlicher Entwicklungsschritte (oder Versionen) von $stuff
- Üblicherweise Trennung zwischen
  - Versionsinformation / Metadaten – sog. Repository
  - Zu versionierenden Daten – sog. Working Tree
- Gibt's schon ewig
- Manuelle Versionsverwaltung: ThesisV1.odt, ThesisV2.odt, ThesisV3.odt, …

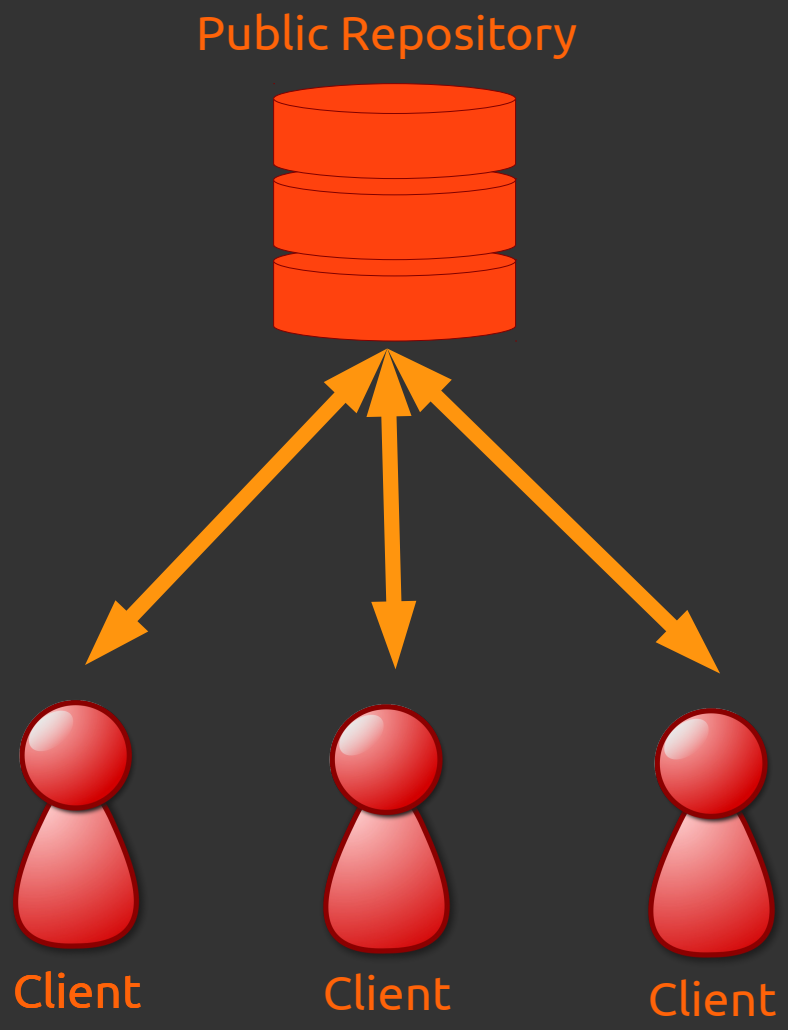
Warum Versionsverwaltung?

- Sicherung von Zwischenschritten / Backup
- Dokumentation des Herstellungsprozesses
- Einfaches Experimentieren durch Branching
- Mehr Metainformationen: Wann wurde was warum geändert?
- Ermöglichung effektiver Zusammenarbeit ohne Datenverlust (A und B schreiben nacheinander auf eine Datei in einem Shared Medium – letzter Schreibvorgang gewinnt)

Workflows

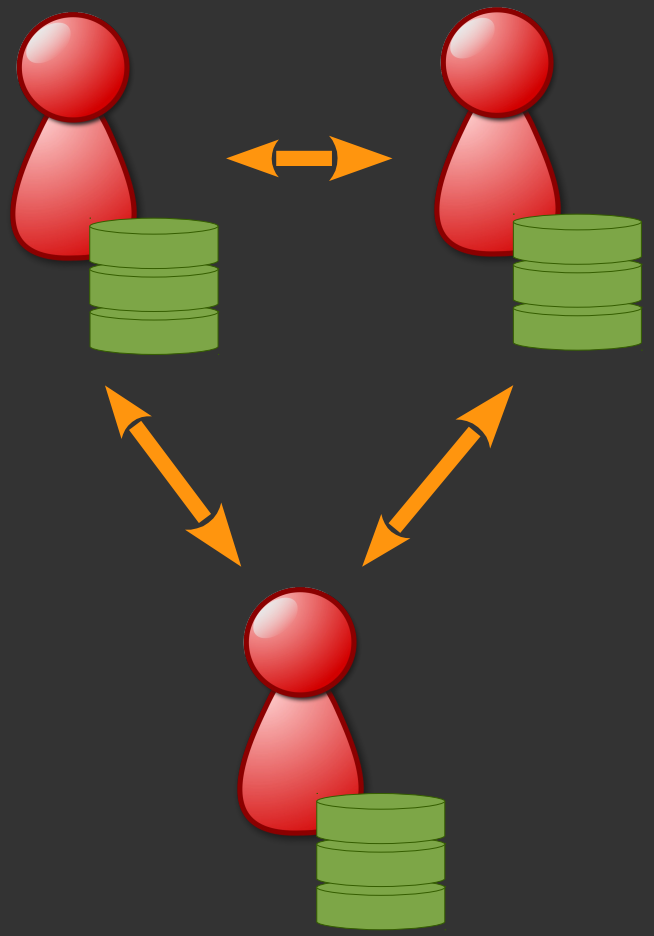
Zentrale Versionsverwaltung

- Ein zentraler Server (genauer: ein zentrales Repository)
- Client speichert keine Versionierungsdaten
- Rein zentralistisches Konzept
- Clients haben Schreib- und Leseberechtigung auf dem Repository



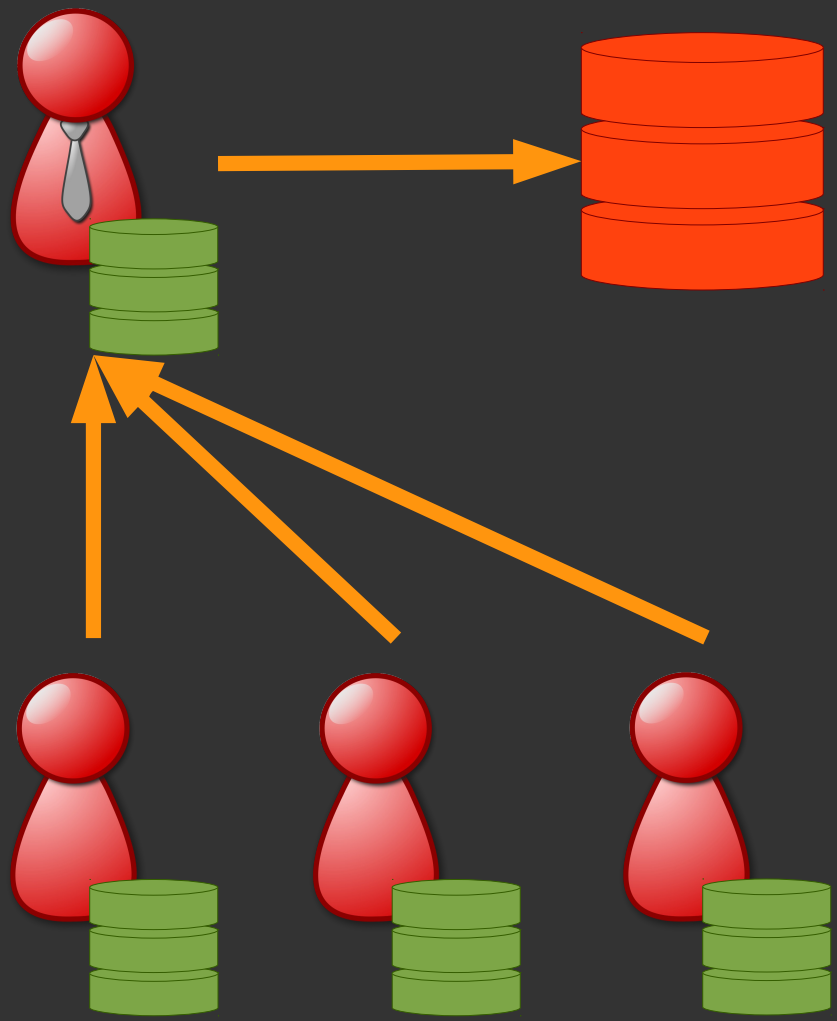
Dezentrale Versionsverwaltung Anarchie

- Jeder Client pflegt eigenes Repository
- Clients haben nur auf eigenes Repo Schreibzugriff
- Austausch von Daten über
  - Pull-Requests (bei öffentlichen Repositories)
  - Patchsets (bei privaten Repositories)
- Absolut Dezentrale Entwicklung



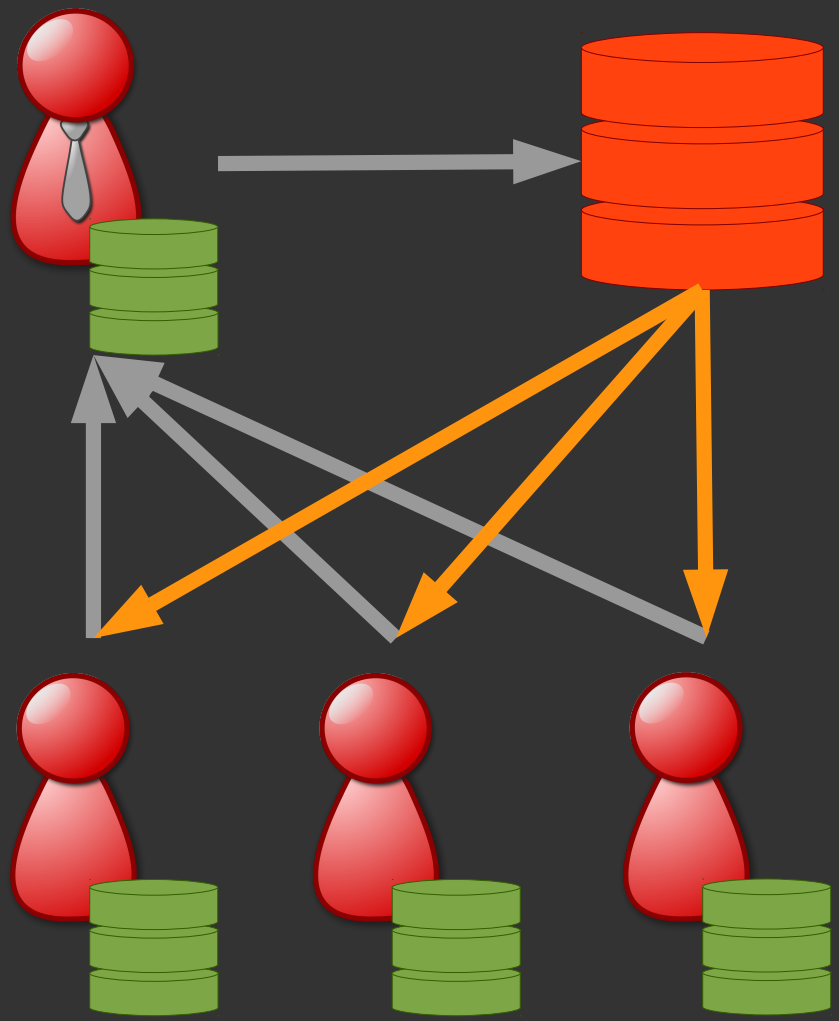
Dezentrale Versionsverwaltung: Gemeinsamer Maintree

- Gemeinsames Repository für die Hauptentwicklung
- Üblicherweise von wenigen Personen betreut, die Code einpflegen („Integration Manager“) - nur diese haben Schreibzugriff auf das Gemeinsame Repository
- Abgesegneter Code wird veröffentlicht



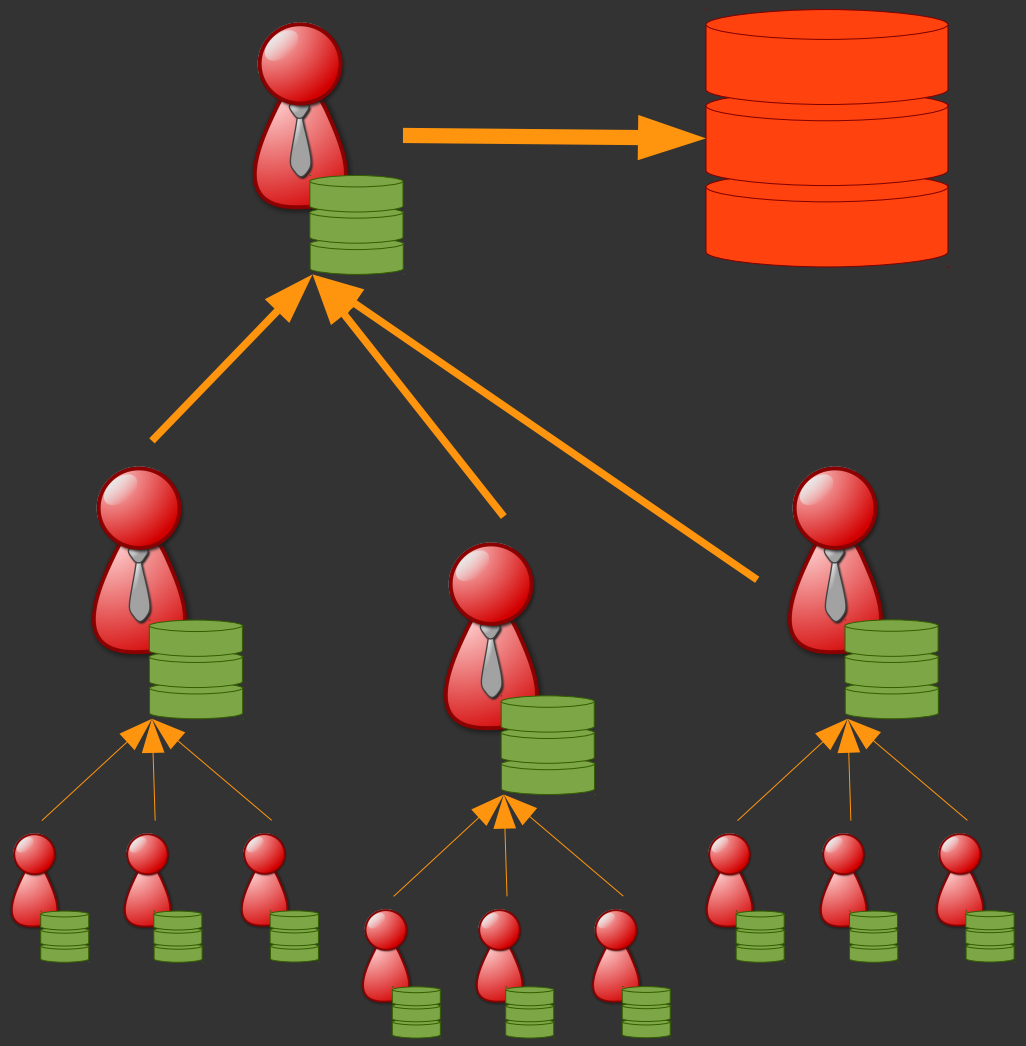
Dezentrale Versionsverwaltung: Gemeinsamer Maintree

- Gemeinsames Repository für die Hauptentwicklung
- Üblicherweise von wenigen Personen betreut, die Code einpflegen („Integration Manager“) - nur diese haben Schreibzugriff auf das Gemeinsame Repository
- Abgesegneter Code wird veröffentlicht
- Clients ziehen Updates aus dem Gemeinsamen Repository



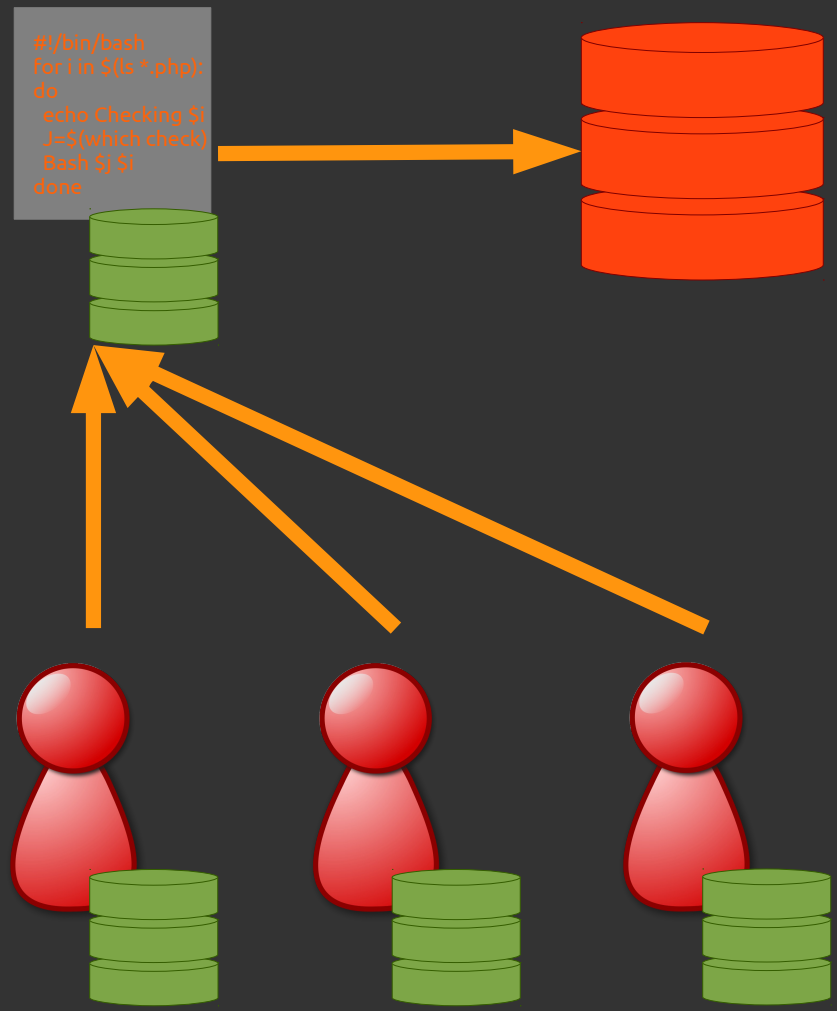
Dezentrale Versionsverwaltung: Gestaffeltes gemeinsames Repo

- Hierarchie aus „Integration Managern“ pflegt jeweils einen Teil
- Verteilung der Verantwortlichkeiten und Lasten auf mehrere Schultern
- Beispielsweise beim Linux Kernel eingesetzt



Dezentrale Versionsverwaltung: Variante: Programm-Gatekeeper

- Gemeinsames Repository für die Hauptentwicklung
- Automatischer Test von Code auf Korrektheit (z.B. durch Test Framework oder Ablehnung von Code mit @ToDo-Tags)



GIT

GIT

- Kann alle Workflows abbilden
- Weite Verbreitung (*this just in: Microsoft baut GIT im Team Foundation Server ein*)
- Verfügbar auf allen großen Plattformen
- Ursprünglich von Linus Torvalds für das Development des Linux Kernels entwickelt
  - Hohe Sicherheit
  - Erprobte Herangehensweise

GIT QuickStart

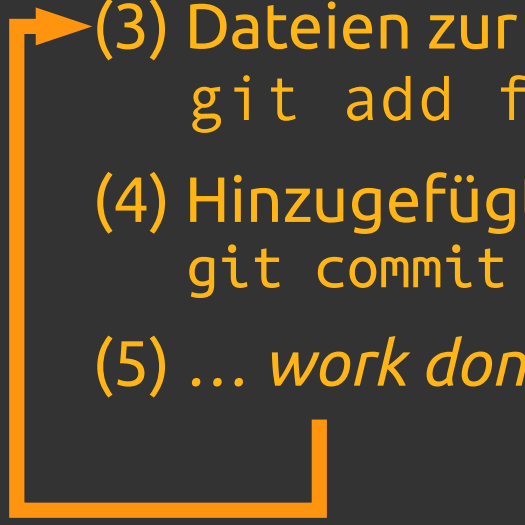
(1) GIT installieren

(2) Repository im aktuellen Verzeichnis anlegen:
```
git init
```

(3) Dateien zur Verwaltung hinzufügen:
```
git add fileA, fileB, dirX/*
```

(4) Hinzugefügte Dateien einpflegen (committen):
```
git commit -m „My commit Message“
```

(5) … *work done here* …



GIT QuickStart

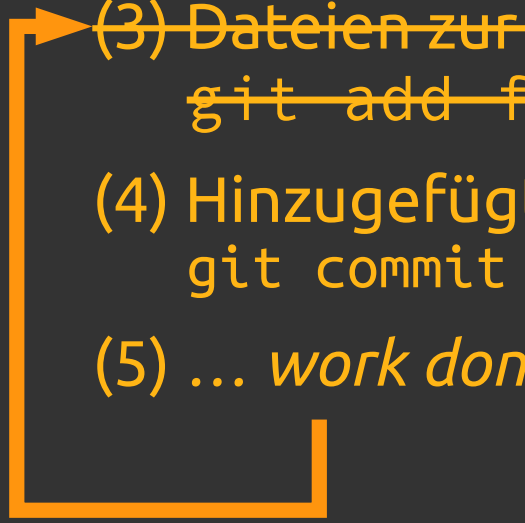
(1) GIT installieren

(2) Repository im aktuellen Verzeichnis anlegen:
```
git init
```

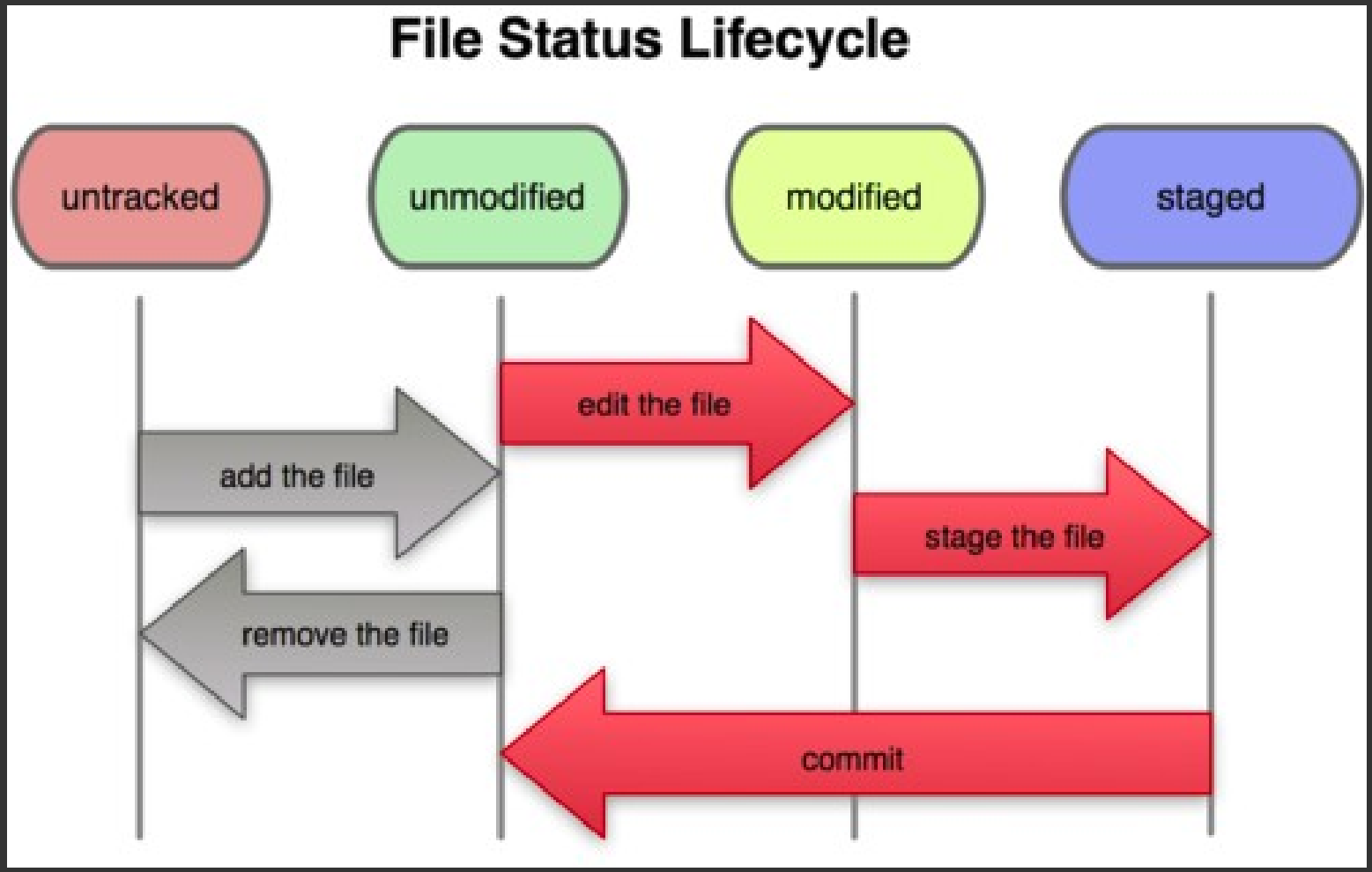
~~(3) Dateien zur Verwaltung hinzufügen:~~
~~`git add fileA, fileB, dirX/*`~~

(4) Hinzugefügte Dateien einpflegen (committen):
```
git commit -a -m „My commit Message“
```

(5) … *work done here* …



GIT QuickStart



Quelle: http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository

Generelle Nutzung

- Hilfe
  ```
  git --help
  git <befehl> --help
  ```
- Status der aktuellen Bearbeitung
  ```
  git status
  ```
- Log anschauen
  ```
  git log
  ```
- Commits sind SHA1-hashsums
  - können zusätzlich per GPG signiert werden
  - Kurzangabe möglich (TAB-Autocomplete auch)

Branching

- Branches sind leichtgewichtige Kopien des aktuellen Bearbeitungsstandes
- Anwendungsgebiete
  - Experimente
  - Getrennte Entwicklung von einzelnen Modulen
  - Trennung unterschiedlicher Versionen (z.B. Stable vs. Development-Branch)
- Automatischer Standardbranch: master (wird beim ersten commit angelegt)

Branching

- Branches Anzeigen
  `git branch`
- Neuen Branch erstellen
  `git branch featureX`
- Zwischen Branches wechseln
  `git checkout featureX`
- *… work done here … committing …*
- Branches zusammenführen
  `git checkout master`
  `git merge featureX`
- Branch löschen
  `git branch -d featureX`

Tagging

- Bezeichnen eines Commits mit einem Namen
- Beispielsweise für Versionierung
- Die meisten Befehle funktionieren auch mit Tags anstatt Commit-Hashtags
- Tag zuweisen
  `git tag Version_1`

Zusammenarbeit

- Vorstellungsrunde
  ```
  git config -global user.name „Raum Zeit“
  git config --global user.email „rzl@example.com“
  ```
- Ein Repository klonen
  ```
  git clone git://...
  ```
- *… arbeiten … committen …*
- Änderungen zurück ins Repo pushen
  ```
  git pull
  git push origin master
  ```
- Einen Patch erstellen
  ```
  git format-patch master --stdout > sendme.patch
  ```
- Einen Patch versenden
  ```
  git send-email --to „John Doe <j@example.com>“ *.patch
  ```

Konfliktlösung

- Was geschieht, wenn sich Änderungen widersprechen (z.B. Änderungen in derselben Funktion)?
- GIT verhindert ein commit und gibt eine Konfliktnachricht beim `git pull` aus.
- GIT kann einen Merge-Konflikt automatisch lösen
- Sonst: Konflikt manuell lösen und committen

```
git checkout --ours -- conflictFile
git commit
git push origin master
```

Hilfreiche Tools

- **GitK**
  Graphische Anzeige von Logs eines Repos
- **TortoiseGIT**
  Windows-Software für den Explorer
- **Meld**
  Sehr gutes Quellcode-Vergleichstool
- **TIG**
  Text-Mode Interface for Git – bunt und in Farbe
- **GIT FTP**
  Nettes Modul zum Veröffentlichen von Working Trees (z.B. für Design oder Homepages)

Mehr GIT?

- Scott Chacon: „Pro GIT“
  http://book.git-scm.com/
- Raumzeitlabor@**GitHub**
  https://github.com/raumzeitlabor
- GIT CheatSheet
  http://jan-krueger.net/development/git-cheat-sheet-extended-edition
- Ask-A-Raumzeitlaborant
- Workshop wurde bereits angefragt → Termin wird nach Rücksprache mit meiner höheren und der tieferen Macht bekannt gegeben

Hackernachwuchs!

$ git bisect good
$ git bisect bad
$ git blame

(Annotations and Blames welcome)

Disclaimer:

Nein, ich weiß nicht, ob es GIT, git, Git oder gi+ heißt.