git & git-flow

Jens Sandmann

Warpzone Münster e.V.

14.12.2013

Überblick

Was ist ein Version Controll System(VCS)?

Hauptaufgaben

- Protokollieren von Änderungen
- Nachvollziehen Wer Wann Was geändert hat
- Gemeinsamer Zugriff auf dieselben Dateien
- Wiederherstellen von älteren Versionen einer Datei

Formen von Versionsverwaltung

Lokale Versionsverwaltung

Es wird nur lokal versioniert. Beispiel: RCS(1982)

Zentrale Versionsverwaltung

Es gibt ein Zentrales Repository *the Truth* mit dem alle Arbeiten.
Erlauben das mehrere Leute gleichzeitig an einem Projekt Arbeiten.
Beispiele: CVS(1990), Subversion(2000)

Verteilte Versionsverwaltung

Verteile Versionsverwaltung

Jeder Nutzer hat Lokal eine vollständige Kopie des gesamten Repositorys. Es kann Zentrale Server geben muss aber nicht.
Beispiele: git, mercury

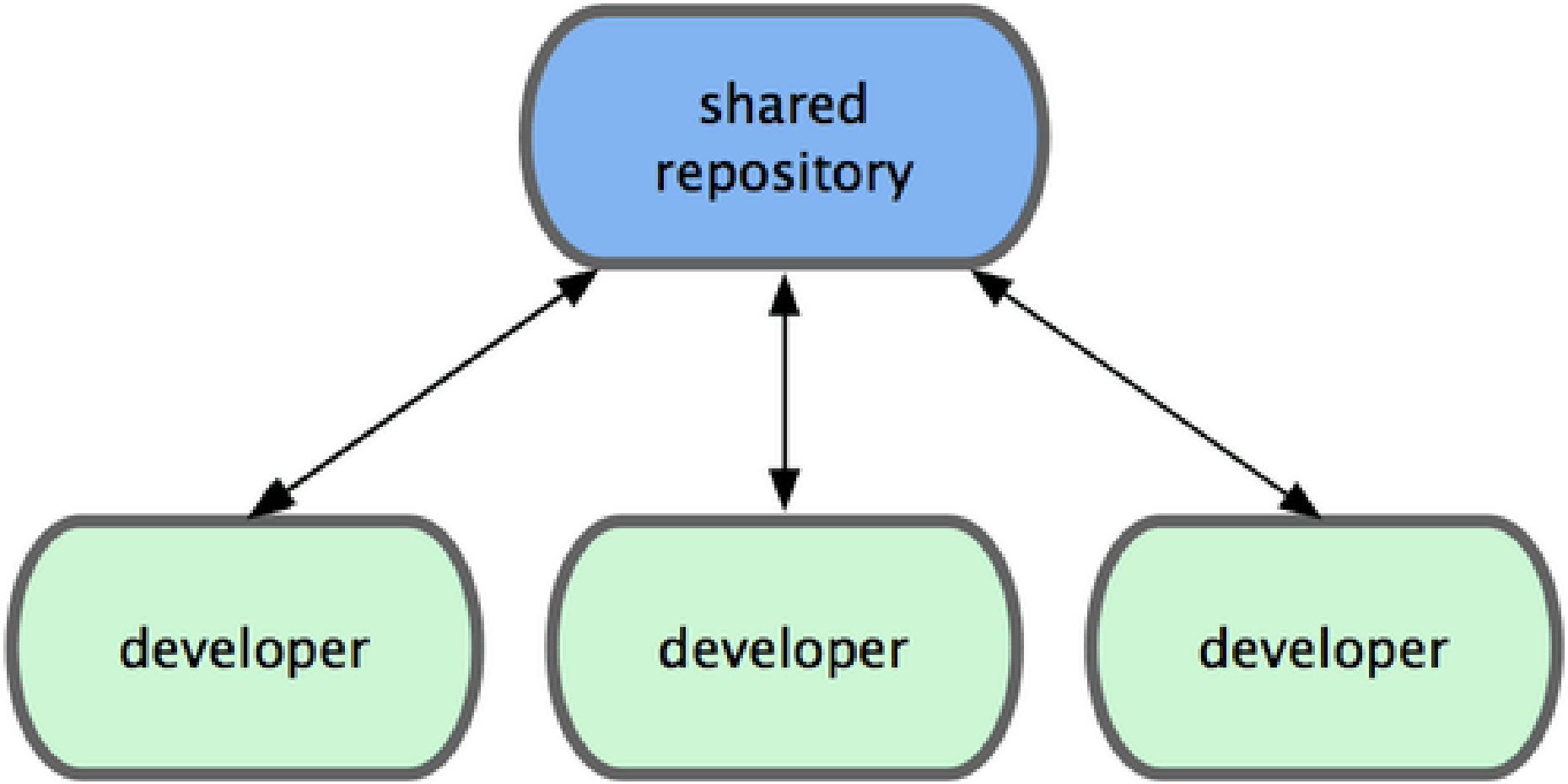
Eigenschaften von git

Ziele von Torvalds

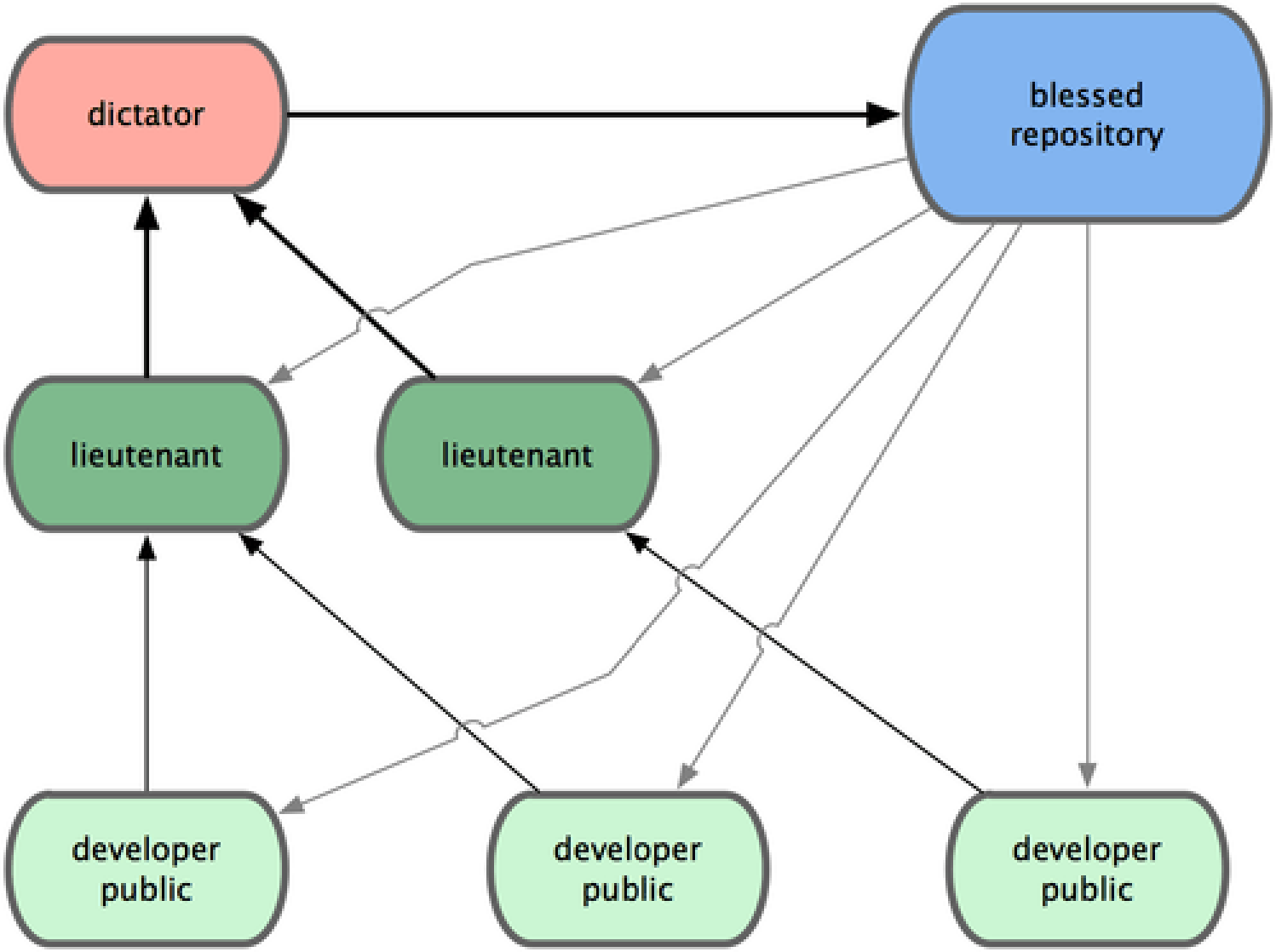
Git wurde Ursprünglich von Linus Torvalds entwickelt, dabei wollte er etwas das besonders folgende Anforderungen erfüllt:

- Unterstützung verteilter Arbeitsabläufe
- Hohe Sicherheit gegen unbeabsichtigte/böswillige Verfälschung
- Hohe Effizienz

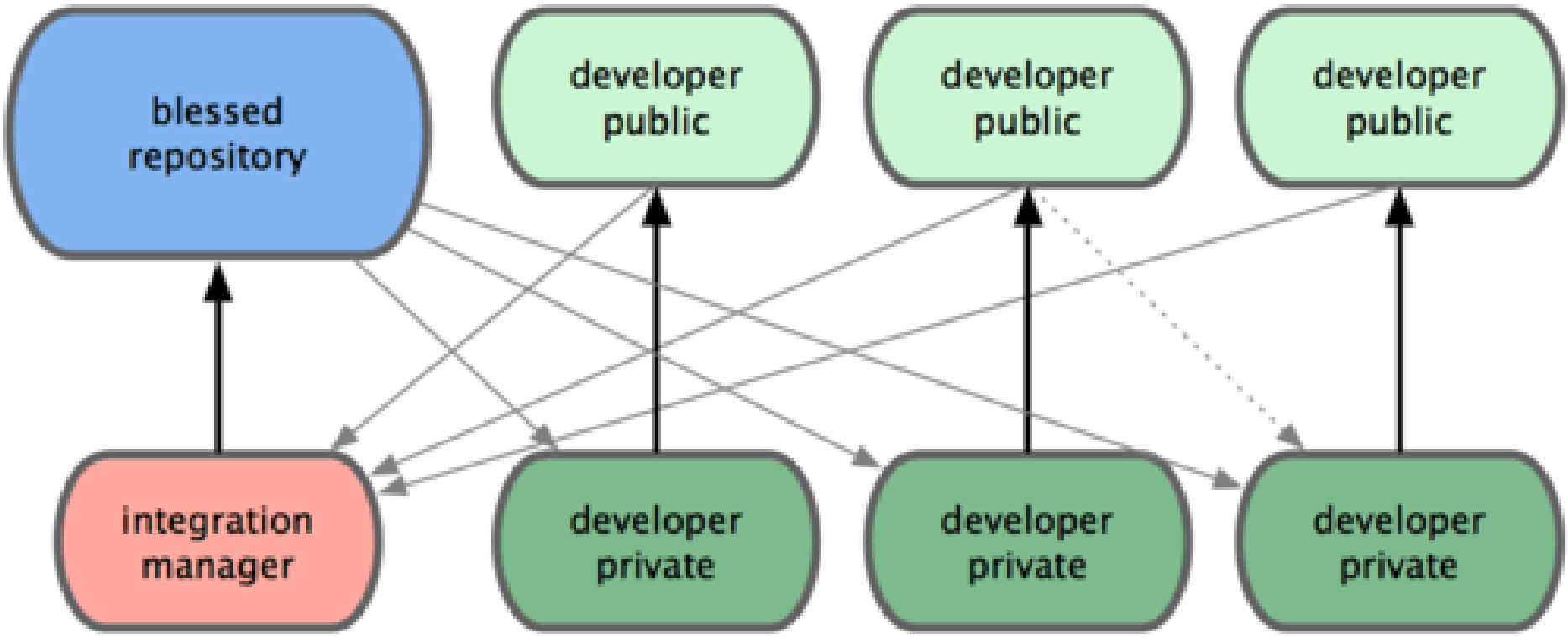
Zentralisierter Workflow



Diktator und Lieutenants (Bsp: Linux)



Integration Manager (Bsp: Github)



Verwendete Begriffe I

Glossar

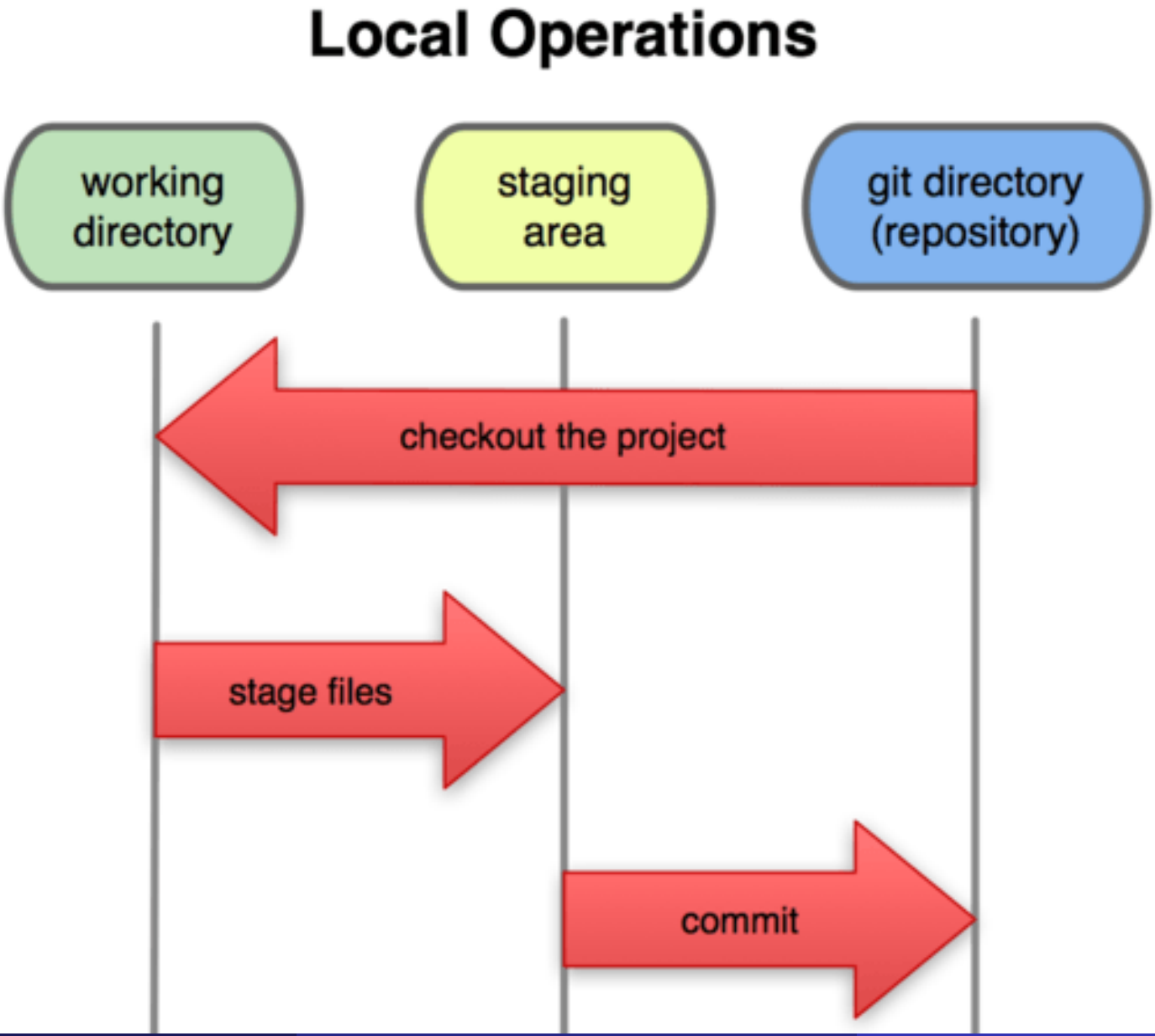
- commit - Ein bestimmter Änderung bzw Stand von Dateien(auch Revision)
- branch - Ein "Zweig", eine Verzweigung von einem Ursprünglichem gleich Stand
- merge - das Zusammenfügen(eng: mergen) von Unterschiedlichen Dateien/Branches/..
- Repository - eng: Lager, Depot, Quelle auf deutsch auch Projektarchiv genannt. Der Ort in dem die Verwalteten Daten liegen

Verwendete Begriffe II

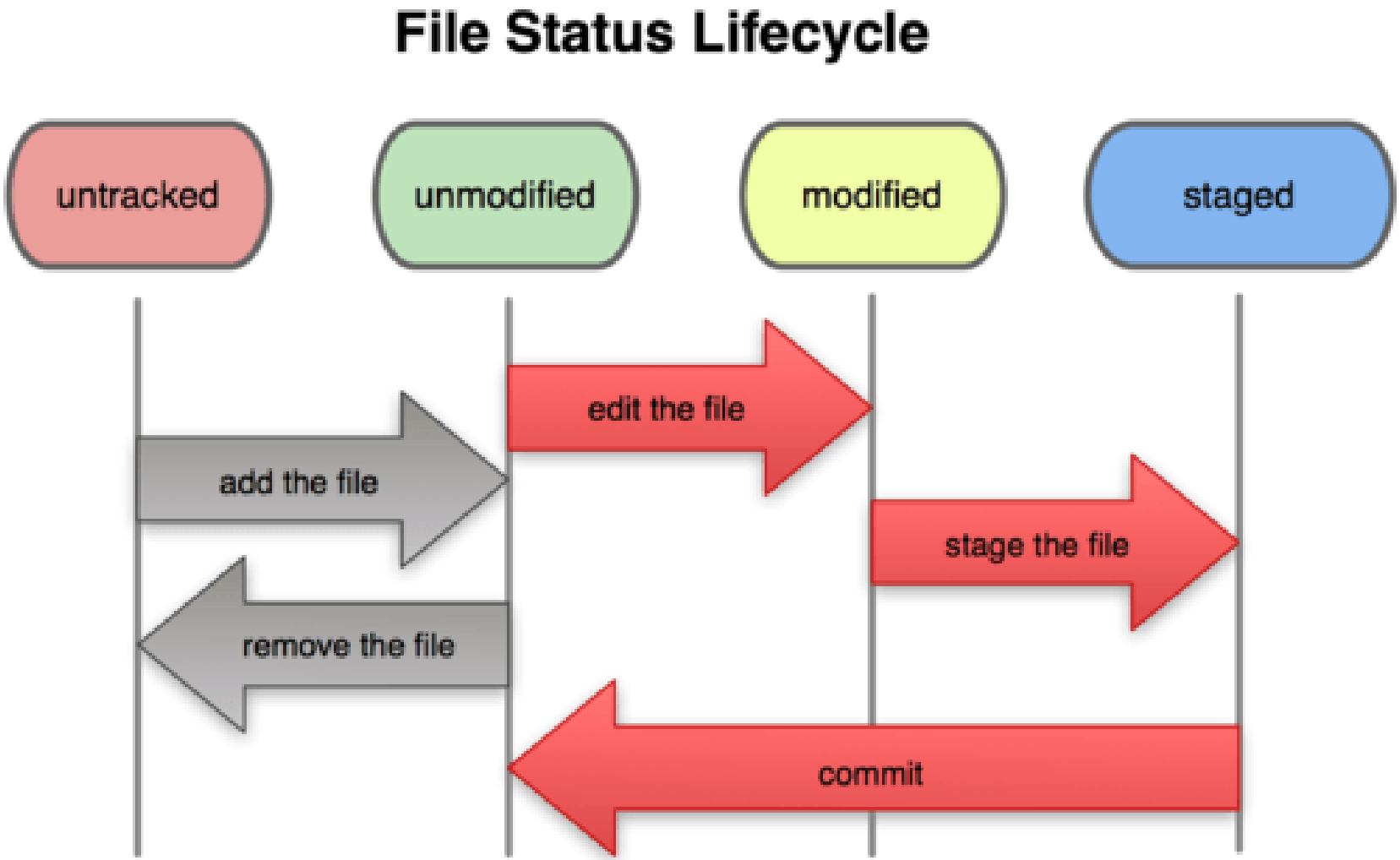
Glosar

- checkout - Zu einen bestimmten branch oder einen bestimmten stand eines branches wechseln
- working directory - Das Arbeitsverzeichnis, hier werden Änderungen vorgenommen.
- stage area - Dateien die für einen commit vorgesehen sind landen in der staging area

Workflow I


Local Operations
working directory
staging area
git directory (repository)
checkout the project
stage files
commit

Workflow II


File Status Lifecycle
untracked
unmodified
modified
staged
edit the file
add the file
stage the file
remove the file
commit

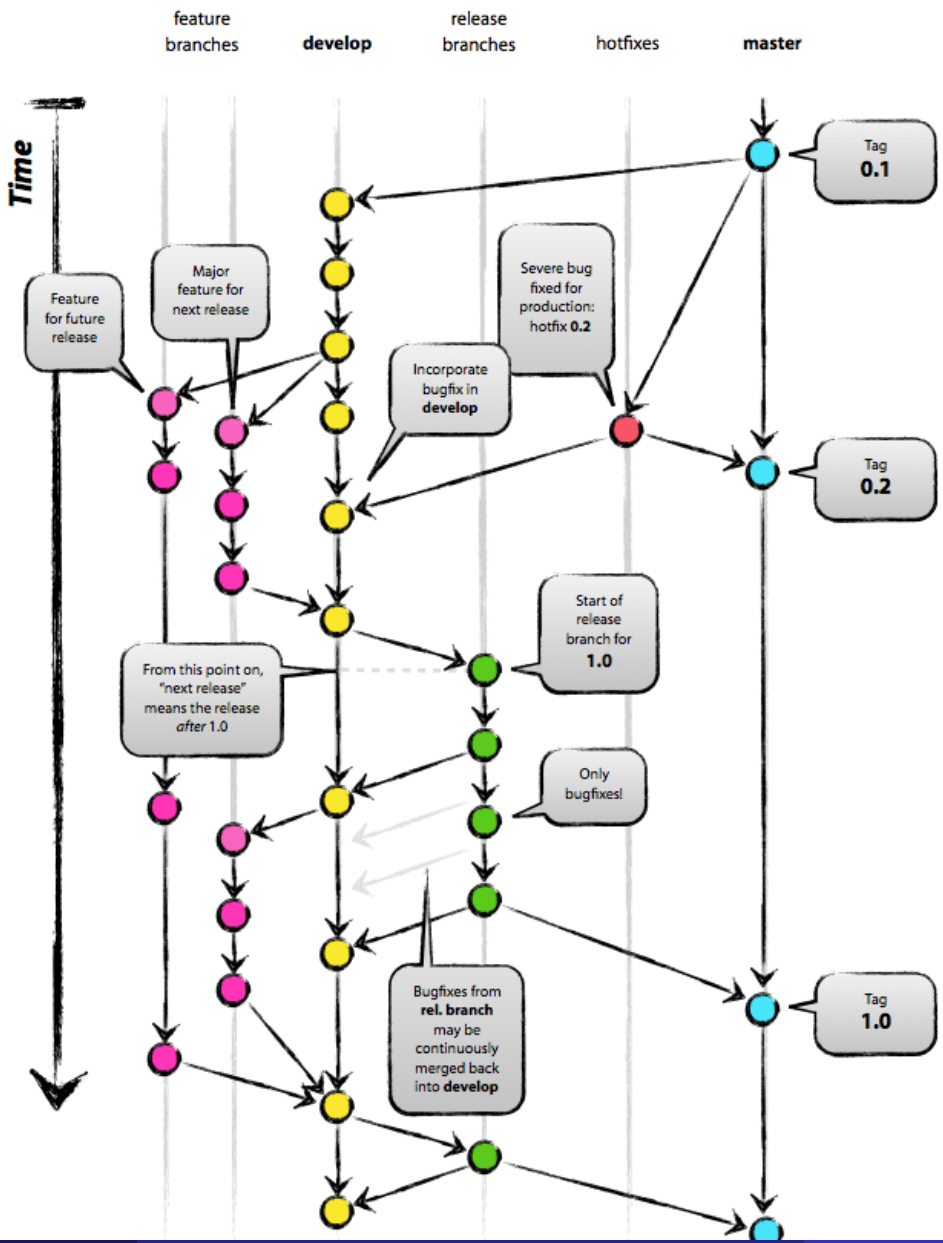
git flow

A successful Git branching model

Die Idee von Vincent Driessen

- Es gibt ein zentrales Repository
- Es gibt zwei Hauptbranches: Main und Develop
- Es gibt verschiedene Unterstützende Branches

git-flow Workflow



Hauptbranches

Main Branch

Der Hauptzweig. Jede Version des Main Branches ist immer Lauffähig und mit Tag versehen so das man gut eine Bestimmte Version auschecken kann.

Develop Branch

Der Hauptentwicklungszweig. Auf diesem findet die Entwicklung statt, es wird aber nicht direkt auf Develop gearbeitet sondern man nutzt dafür die Unterstützendes Branches. Develop sollte Funktionstüchtig sein kann aber noch Bugs enthalten. Von diesem Branch aus würden Nightly Build erstellt.

Unterstützende Branches

Feature Branch

Die Feature Branches gehen vom Develop aus. Es wird immer genau ein Feature Entwickelt und nach Abschluss wieder in Develop gemerged. Ein Feature Branch lebt nur so lange wie bis das Feature fertig gestellt wurde, danach wird er gelöscht.

Unterstützendes Branches

Release Branch

Ein Feature branch geht auch von Develop aus, und wird am Ende wieder in Develop und Main gemerged. Das Ziel ist ein neues Release. Es muss sichergestellt sein das alle angestrebten Features für das neue Release in Develop sind, Features für die neue Version aber noch nicht eingepflegt wurden.

Unterstützendes Branches

Hotfix Branch

Hotfix Branches sind die einzigen die von Master ausgehen. Sie sind erforderlich wenn im Main Branch ein Fehler gefunden wurde. Es wird immer nur der Spezielle Bug gefixt. Änderungen aus Hotfix Branches landen sowohl in Main als auch in Develop, damit es keine Regressionen gibt.

Zusammenfassung

Zusammenfassung

- Die Hauptbranches leben ständig.
- Die Unterstützenden Branches leben nur so lange bis sie abgeschlossen sind.
- Merges werden mit der –no-ff Option vorgenommen so das ersichtlich ist welche Commits aus einem Feature Branch stammen, und es später nicht als ein linearer Strang erscheint.

git nutzen

Git einrichten

Grundlegende Einstellungen

Alle Befehle von git starten mit dem *git* Befehl. Die eigene Identität

```
git config user.name "Jens Sandmann"
git config user.email jens@member.warpzone.ms
```

Mit dem Parameter –global als Standardeinstellungen für den User, ansonsten gilt das pro Repository

Ein Repository anlegen

Init

Die einfachste Möglichkeit ein git repository zu erstellen ist einfach auf der Kommandozeile folgendes einzugeben

```
git init
```

Dadurch wird das aktuelle Verzeichnis zu einem git leeren(!) git Repository mit dem man Arbeiten kann.

Ein bestehendes Repository clonen

Clone

Alternativ kann man ein schon bestehendes git Repository z.b. von git-hub oder unserem git clonen

```
git clone git@warpzone.ms:git-workshop.git
```

Dabei wird Unterhalb des Bestehendes Verzeichnisses ein neues Verzeichnis mit den Namen des git Repositorys angelegt in dem sich die Dateien befinden.

Stand eines Verzeichnisses

Aktueller Status

Den Aktuellen Status eines git Verzeichnisses lässt man sich per

```
git status
```

ausgeben

Geschichte eines Verzeichnisses

Die Geschichte eines Verzeichnisses lässt sich per *log* anzeigen

```
git log
```

Über den *-p* Parameter kann man sich die Änderungen im Quelltext anzeigen lassen.

Änderungen nachvollziehen

Anzeigen von Änderungen

Welche Dateien in der Working Area verändert wurden zeigt uns *git status* an, aber erst per *git diff* sehen wir welche Änderungen genau an den einzelnen Dateien vorgenommen wurden. Das gleiche funktioniert auch für die Dateien in der Staging Area per *–cache* Parameter

```
git diff
git diff --cached
```

Einen Commit erstellen

Dateien zu einem Commit hinzufügen

Dateien müssen erst in die Staging Area hinzugefügt werden damit sie Teil des Commits werden.

```
git add test1.txt
git add *.txt
git add img/
```

Einen Commit erzeugen

Commiten

Das eigentliche Commiten erzeugt aus allen Dateien&Änderungen die gestaged sind einen Commit. Am besten lässt man sich diese vorher nochmal anzeigen.

```
git status
git commit
```

Anschließend die Commit Message im Editor eingeben. Alternativ:

```
git commit -m "Commit Message"
```

Arbeiten mit Entfernten Repositorys

Änderungen erhalten

Wenn man mit entfernten, sogenannten Remotes, arbeitet, kann und will man normalerweise seine Änderungen zurück an diese senden und Änderungen von anderen auf dem Remote auch erhalten.

```
git fetch <remote branch> <lokaler branch>
git merge
git pull <remote branch> <lokaler branch>
```

Bei *fetch* und *merge* hat man mehr Kontrolle. Bei *pull* macht git alle Arbeit, solange es keine merge Konflikte gibt.

Branches

Branches erzeugen und anzeigen

Ein Branch wird immer von dem Branch aus erzeugt auf dem man sich gerade befindet. Wenn man also verschiedene Branches hat sollte man sicherstellen das man vom richtigen aus branched.

```
git branch <name des neuen branches>
git branch -a # zeigt alle branches an
git checkout -b <name des neuen branches>
git checkout <name des bestehendes branches>
```

Quellen

- http://www.git-scm.com/book/en/ Die Quelle für alle Fragen um git
- Git in der deutschen Wikipeida
- A successful Git branching model
- git-flow cheatsheet