Quelle: `http://git-scm.com/downloads/logos`

GIT @ MMS, GRUNDKONZEPTE

Branching, Merging, Repositorystrukturen und mehr

AUFBAU
TEIL I: GIT ALLGEMEIN

Aufbau von Teil I

Basiswissen

Branches, Tags, Merge und Rebase

Repository-Organisation

Hooks

Weiterführendes

AUFBAU

TEIL II: GIT IN DER MMS

Aufbau von Teil II

Unterstützung für Projekte

MMS-Tools

Empfohlene Branching-Strukturen

Einbindung in die Toolkette

Teil I

GIT ALLGEMEIN

BASISWISSEN

Begriffe
Installation und Ersteinrichtung
Basiskommandos
Revision-Spezifikation

Elemente von Git-Repositories

- Durch SHA1-Hash identifizierbar:
  - Blob (Datei)
  - Tree
  - Commit

Elemente von Git-Repositories

- Durch SHA1-Hash identifizierbar:
  - Blob (Datei)
  - Tree
  - Commit
- Verweise auf Commits:
  - Branches
  - Tags

Elemente von Git-Repositories

- Durch SHA1-Hash identifizierbar:
  - Blob (Datei)
  - Tree
  - Commit
- Verweise auf Commits:
  - Branches
  - Tags
- Sonstiges:
  - Hooks (Skripte)
  - .git/config
  - Remotes
  - Gitlinks (für Submodule)

Trees und Commits

Trees in Repositories

- Working Copy (Checkout)
- Index (Vorbereitung für nächsten Commit)
- HEAD (letzter Commit im aktuellen Branch)

Trees und Commits

Trees in Repositories

- Working Copy (Checkout)
- Index (Vorbereitung für nächsten Commit)
- HEAD (letzter Commit im aktuellen Branch)

Commits

- gerichteter azyklischer Graph (directed acyclic graph, DAG)
- jeder Commit (außer initialem Commit) hat 1-n Parents
- Commits sind Referenzen auf einen Tree plus Metadaten (Author, Zeitstempel, Kommentar, ...)

Grundeinrichtung

- git installieren

  Windows: Installation mit cygwin [1] setup.exe oder Git for Windows [2] (früher msysgit)

  MacOS-X: Installation mit dem Mac-OS-Installer [3]

  RedHat: `yum install git`

  Debian: `aptitude install git`

Grundeinrichtung

- git installieren

  Windows: Installation mit cygwin [1] setup.exe oder Git for Windows [2] (früher msysgit)

  MacOS-X: Installation mit dem Mac-OS-Installer [3]

  RedHat: `yum install git`

  Debian: `aptitude install git`

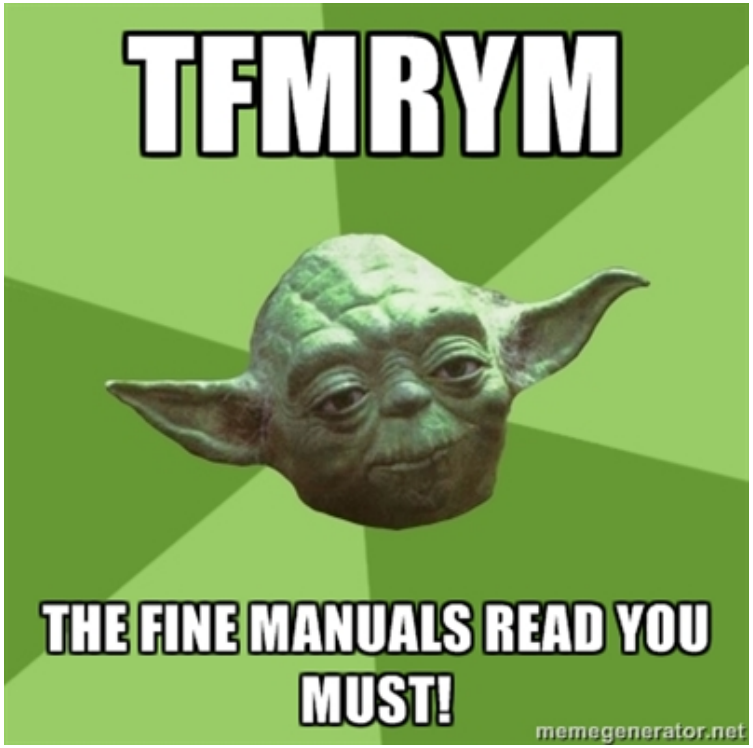
- Wer bin ich?

```
git config --global user.name "Jan␣Dittberner"
git config --global user.email ➘
     jan.dittberner@t-systems.com
```

BASISWISSEN
BASISKOMMANDOS

Use the docs!

```
git help <command>
```



Repository anlegen

- neues lokales Entwicklungsrepository:
  `git init <repo>`

Repository anlegen

- neues lokales Entwicklungsrepository:
  `git init <repo>`
- neues lokales Bare-Repository:
  `git init --bare <repo.git>`

Repository anlegen

- neues lokales Entwicklungsrepository:
  `git init <repo>`
- neues lokales Bare-Repository:
  `git init --bare <repo.git>`
- bestehendes Repository clonen:
  `git clone <repourl> [<target>]`

lokales Arbeiten

- Änderungen zum Index hinzufügen:
  `git add [<file> …]`

lokales Arbeiten

- Änderungen zum Index hinzufügen:
  `git add [<file> …]`
- Dateien löschen:
  `git rm [<file> …]`

lokales Arbeiten

- Änderungen zum Index hinzufügen:
  `git add [<file> …]`
- Dateien löschen:
  `git rm [<file> …]`
- Aktuellen Status von Index und Working Copy:
  `git status`

lokales Arbeiten

- Änderungen zum Index hinzufügen:
  `git add [<file> …]`
- Dateien löschen:
  `git rm [<file> …]`
- Aktuellen Status von Index und Working Copy:
  `git status`
- Informationen zu Objekt:
  `git show <revname>` (später bei Revision-Spezifikation mehr dazu)

lokales Arbeiten

- Änderungen zum Index hinzufügen:
  `git add [<file> …]`
- Dateien löschen:
  `git rm [<file> …]`
- Aktuellen Status von Index und Working Copy:
  `git status`
- Informationen zu Objekt:
  `git show <revname>` (später bei Revision-Spezifikation mehr dazu)
- Unterschied zwischen Index und Working Copy:
  `git diff`

History ansehen

- Log für aktuellen Branch:
  `git log`

History ansehen

- Log für aktuellen Branch:

```
git log
```

- Tipp:

```
git config --global alias.lga "log␣--graph␣↵
     --decorate␣--oneline␣--color␣--all"
git lga
```

History ansehen

- Log für aktuellen Branch:

```
git log
```

- Tipp:

```
git config --global alias.lga "log␣--graph␣↵
     --decorate␣--oneline␣--color␣--all"
git lga
```

- Letzte Änderungen am lokalen Repository:

```
git reflog
```

Selektives Arbeiten mit dem Index

- Commit interaktiv in Index übernehmen:
  `git add -i [<file> …]`

Selektives Arbeiten mit dem Index

- Commit interaktiv in Index übernehmen:
  `git add -i [<file> …]`
- Dateien patch-weise in Index übernehmen:
  `git add -p [<file> …]`

Selektives Arbeiten mit dem Index

- Commit interaktiv in Index übernehmen:
  `git add -i [<file> …]`
- Dateien patch-weise in Index übernehmen:
  `git add -p [<file> …]`
- nur Dateien in Index aufnehmen, die schon getrackt werden, gelöschte Dateien aus Index entfernen:
  `git add -u [<file> …]`

Aktionen rückgängig machen

- Dateienzustand in Index verwerfen:
  `git reset [<file> …]`

Aktionen rückgängig machen

- Dateienzustand in Index verwerfen:
  `git reset [<file> …]`
- Dateien in Working Copy zurücksetzen:
  `git checkout [<file> …]`

Aktionen rückgängig machen

- Dateienzustand in Index verwerfen:
  `git reset [<file> …]`
- Dateien in Working Copy zurücksetzen:
  `git checkout [<file> …]`
- Working Copy verwerfen:
  `git reset --hard HEAD`

Aktionen rückgängig machen

- Dateienzustand in Index verwerfen:
  `git reset [<file> …]`
- Dateien in Working Copy zurücksetzen:
  `git checkout [<file> …]`
- Working Copy verwerfen:
  `git reset --hard HEAD`
- Commit erzeugen, der anderen Commit rückgängig macht:
  `git revert [<commitid>]`

BASISWISSEN
REVISION-SPEZIFIKATION

- `git help rev-parse`

BASISWISSEN
REVISION-SPEZIFIKATION

- `git help rev-parse`
- verwendbar für viele Operationen (log, push/pull, diff, ...)

BASISWISSEN
REVISION-SPEZIFIKATION

- `git help rev-parse`
- verwendbar für viele Operationen (log, push/pull, diff, ...)
- `<sha1>` (auch abgekürzt) für Blobs, Commits, Trees

BASISWISSEN
REVISION-SPEZIFIKATION

- `git help rev-parse`
- verwendbar für viele Operationen (log, push/pull, diff, ...)
- `<sha1>` (auch abgekürzt) für Blobs, Commits, Trees
- `<refname>` z.B.:

BASISWISSEN
REVISION-SPEZIFIKATION

- `git help rev-parse`
- verwendbar für viele Operationen (log, push/pull, diff, ...)
- `<sha1>` (auch abgekürzt) für Blobs, Commits, Trees
- `<refname>` z.B.:
  - `master`, `heads/master`, `refs/heads/master` für lokale Branches

BASISWISSEN
REVISION-SPEZIFIKATION

- `git help rev-parse`
- verwendbar für viele Operationen (log, push/pull, diff, ...)
- `<sha1>` (auch abgekürzt) für Blobs, Commits, Trees
- `<refname>` z.B.:
  - `master`, `heads/master`, `refs/heads/master` für lokale Branches
  - `r1.1`, `refs/tags/r1.1` für Tags

- `git help rev-parse`
- verwendbar für viele Operationen (log, push/pull, diff, ...)
- `<sha1>` (auch abgekürzt) für Blobs, Commits, Trees
- `<refname>` z.B.:
  - `master`, `heads/master`, `refs/heads/master` für lokale Branches
  - `r1.1`, `refs/tags/r1.1` für Tags
  - `refs/remotes/origin/development` für Remote-Tracking-Branches

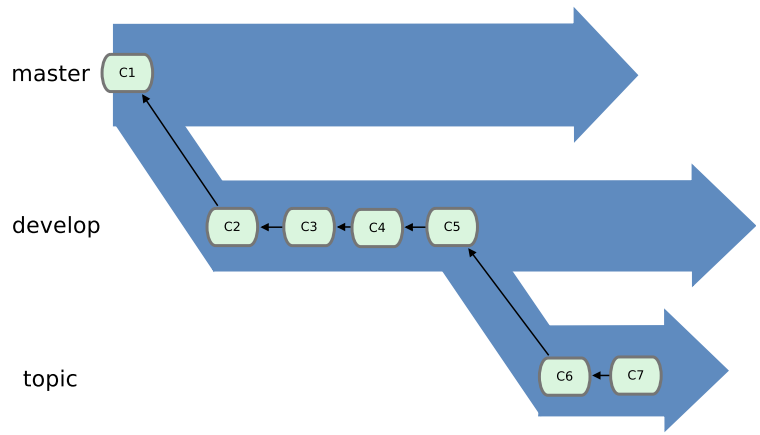
BRANCHES, TAGS, MERGE UND REBASE

Allgemeines zu Branches
Tags
Stash (Git-Kommando)
Branching-Modelle
Merging
Merge-Tools
Patch-Workflow
Cherry-Picking
Rebasing

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

- Branches sind leichtgewichtig

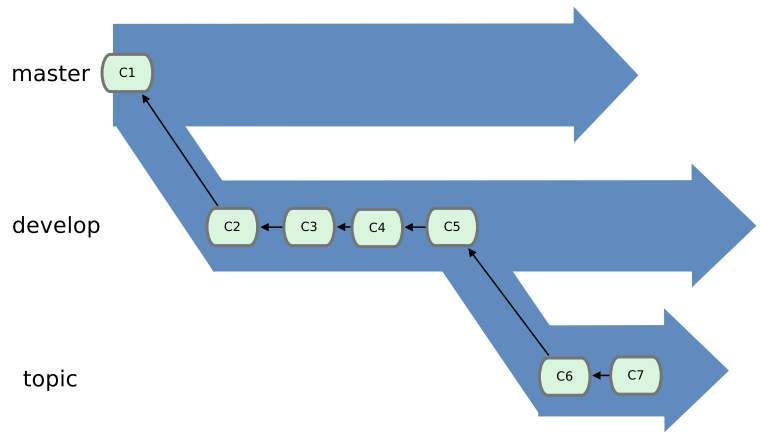


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

- Branches sind leichtgewichtig
- Branches nutzen z.B. für:

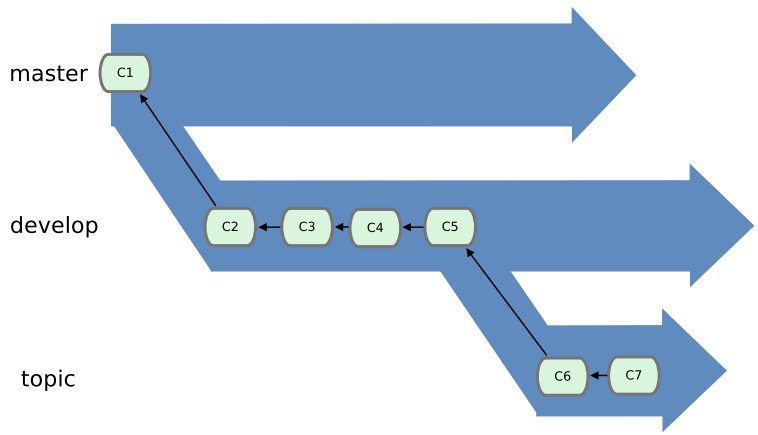


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

- Branches sind leichtgewichtig
- Branches nutzen z.B. für:
  - Ausprobieren von Änderungen/Experimente

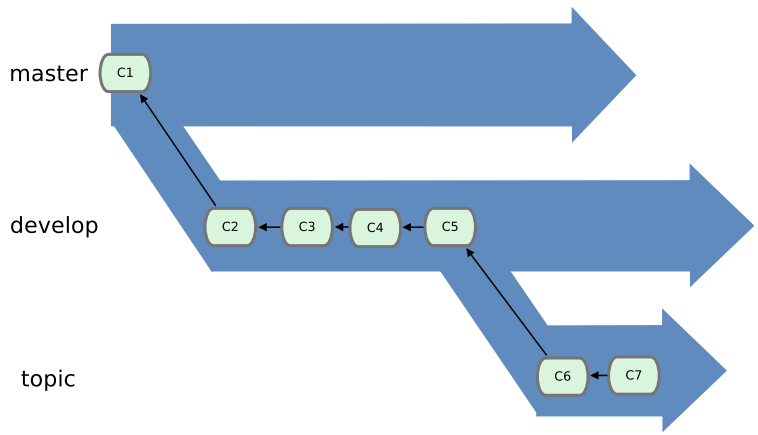


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

- Branches sind leichtgewichtig
- Branches nutzen z.B. für:
  - Ausprobieren von Änderungen/Experimente
  - Isolation von Features

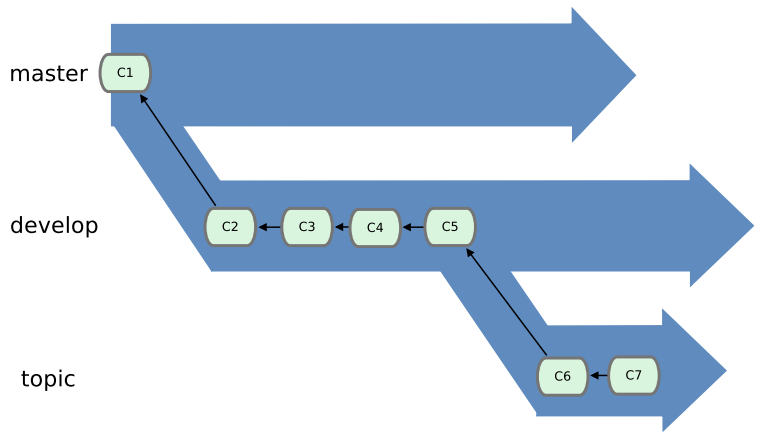


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

- Branches sind leichtgewichtig
- Branches nutzen z.B. für:
  - Ausprobieren von Änderungen/Experimente
  - Isolation von Features
  - Umschalten zwischen Aufgaben

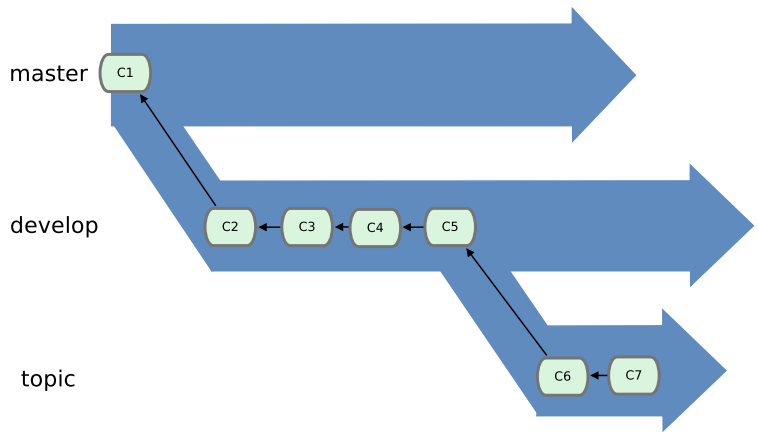


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

- Branches sind leichtgewichtig
- Branches nutzen z.B. für:
  - Ausprobieren von Änderungen/Experimente
  - Isolation von Features
  - Umschalten zwischen Aufgaben
  - Releases

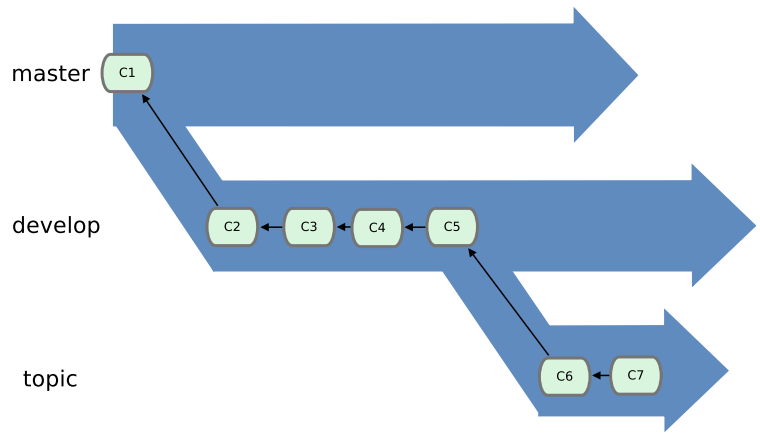


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

- Branches sind leichtgewichtig
- Branches nutzen z.B. für:
  - Ausprobieren von Änderungen/Experimente
  - Isolation von Features
  - Umschalten zwischen Aufgaben
  - Releases
  - Qualitätssicherung

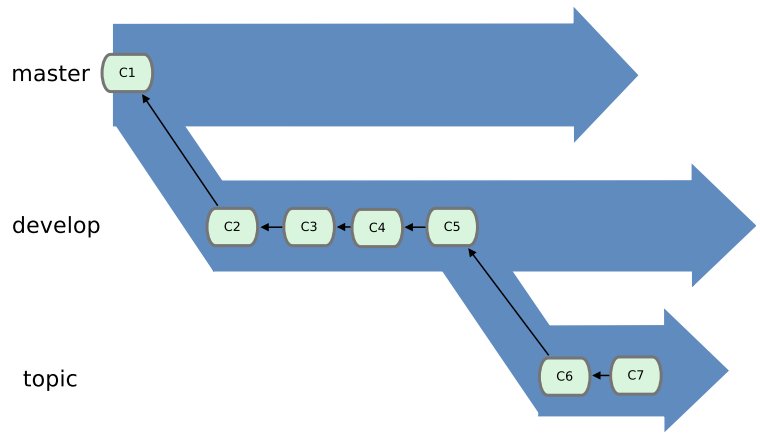


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

- Branches sind leichtgewichtig
- Branches nutzen z.B. für:
  - Ausprobieren von Änderungen/Experimente
  - Isolation von Features
  - Umschalten zwischen Aufgaben
  - Releases
  - Qualitätssicherung
  - Wartungsbranches



Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

Kommandos

```
git branch <branchname>
git checkout <branchname>
git branch -d <branchname>
git branch
git branch -D <branchname>
git show-branch
```

Informationen zu Branches

- Commits in Branch

  ```
  git log <branchname>
  ```

Informationen zu Branches

- Commits in Branch
  `git log <branchname>`
- Commits in development die nicht in master sind:
  `git log master..development`

Informationen zu Branches

- Commits in Branch
  `git log <branchname>`
- Commits in development die nicht in master sind:
  `git log master..development`
- Unterschied zwischen development und master:
  `git diff master..development`

BRANCHES, TAGS, MERGE UND REBASE

ALLGEMEINES ZU BRANCHES

Informationen zu Branches

- Commits in Branch
  `git log <branchname>`
- Commits in development die nicht in master sind:
  `git log master..development`
- Unterschied zwischen development und master:
  `git diff master..development`
- mehr in `git help rev-parse` unter SPECIFYING REVISIONS und SPECIFYING RANGES

BRANCHES, TAGS, MERGE UND REBASE

TAGS

- Markierung/Name für Commit

BRANCHES, TAGS, MERGE UND REBASE

TAGS

- Markierung/Name für Commit
- Anders als bei SVN ist ein Tag kein Branch

BRANCHES, TAGS, MERGE UND REBASE

TAGS

- Markierung/Name für Commit
- Anders als bei SVN ist ein Tag kein Branch
- kann (mit GnuPG/PGP) signiert werden

BRANCHES, TAGS, MERGE UND REBASE

TAGS

- Markierung/Name für Commit
- Anders als bei SVN ist ein Tag kein Branch
- kann (mit GnuPG/PGP) signiert werden
- Verwendung als Marker (z.B. für bestimmte Jenkins-Workflows)

BRANCHES, TAGS, MERGE UND REBASE

TAGS

- Markierung/Name für Commit
- Anders als bei SVN ist ein Tag kein Branch
- kann (mit GnuPG/PGP) signiert werden
- Verwendung als Marker (z.B. für bestimmte Jenkins-Workflows)
- Verwendung für Markierung von Lieferständen/Releases

BRANCHES, TAGS, MERGE UND REBASE

STASH (GIT-KOMMANDO)

- kurzes Zwischenlagern von Änderungen an der Working Copy

BRANCHES, TAGS, MERGE UND REBASE

STASH (GIT-KOMMANDO)

- kurzes Zwischenlagern von Änderungen an der Working Copy
- rein lokal

BRANCHES, TAGS, MERGE UND REBASE

STASH (GIT-KOMMANDO)

- kurzes Zwischenlagern von Änderungen an der Working Copy
- rein lokal
- kann zum Transplantieren in andere Branches genutzt werden

BRANCHES, TAGS, MERGE UND REBASE

STASH (GIT-KOMMANDO)

- kurzes Zwischenlagern von Änderungen an der Working Copy
- rein lokal
- kann zum Transplantieren in andere Branches genutzt werden
- funktioniert wie ein Stack

BRANCHES, TAGS, MERGE UND REBASE

STASH (GIT-KOMMANDO)

- kurzes Zwischenlagern von Änderungen an der Working Copy
- rein lokal
- kann zum Transplantieren in andere Branches genutzt werden
- funktioniert wie ein Stack
- Kommandos:

```
git stash [save]
git stash pop
git stash apply
```

BRANCHES, TAGS, MERGE UND REBASE

BRANCHING-MODELLE

- SVN-like (Entwicklungsbranch + ggf. Release-Branches)

BRANCHES, TAGS, MERGE UND REBASE

BRANCHING-MODELLE

- SVN-like (Entwicklungsbranch + ggf. Release-Branches)
- evtl. + QA-Branch, z.B. für Integration mit CI-Tools wie Jenkins

BRANCHES, TAGS, MERGE UND REBASE

BRANCHING-MODELLE

- SVN-like (Entwicklungsbranch + ggf. Release-Branches)
- evtl. + QA-Branch, z.B. für Integration mit CI-Tools wie Jenkins
- Feature-Branches, User-Branches

BRANCHES, TAGS, MERGE UND REBASE

BRANCHING-MODELLE

- SVN-like (Entwicklungsbranch + ggf. Release-Branches)
- evtl. + QA-Branch, z.B. für Integration mit CI-Tools wie Jenkins
- Feature-Branches, User-Branches
- + Bugfix-Branches

BRANCHES, TAGS, MERGE UND REBASE

BRANCHING-MODELLE

- SVN-like (Entwicklungsbranch + ggf. Release-Branches)
- evtl. + QA-Branch, z.B. für Integration mit CI-Tools wie Jenkins
- Feature-Branches, User-Branches
- + Bugfix-Branches
- ...

BRANCHES, TAGS, MERGE UND REBASE

BRANCHING-MODELLE

git-flow

- Best Practice für Arbeit mit Feature-Branches, Hotfixes, Releases, Tags

git-flow

- Best Practice für Arbeit mit Feature-Branches, Hotfixes, Releases, Tags
- beschrieben in „A successful git branching model“ [5]

git-flow

- Best Practice für Arbeit mit Feature-Branches, Hotfixes, Releases, Tags
- beschrieben in „A successful git branching model“ [5]
- Tool-Unterstützung für shell mit `git-flow` [6]

```
git flow feature
git flow release
```

git-flow

- Best Practice für Arbeit mit Feature-Branches, Hotfixes, Releases, Tags
- beschrieben in „A successful git branching model“ [5]
- Tool-Unterstützung für shell mit `git-flow` [6]

```
git flow feature
git flow release
```

- Unterstützung in Atlassian SourceTree [7]

git-flow

- Best Practice für Arbeit mit Feature-Branches, Hotfixes, Releases, Tags
- beschrieben in „A successful git branching model“ [5]
- Tool-Unterstützung für shell mit `git-flow` [6]

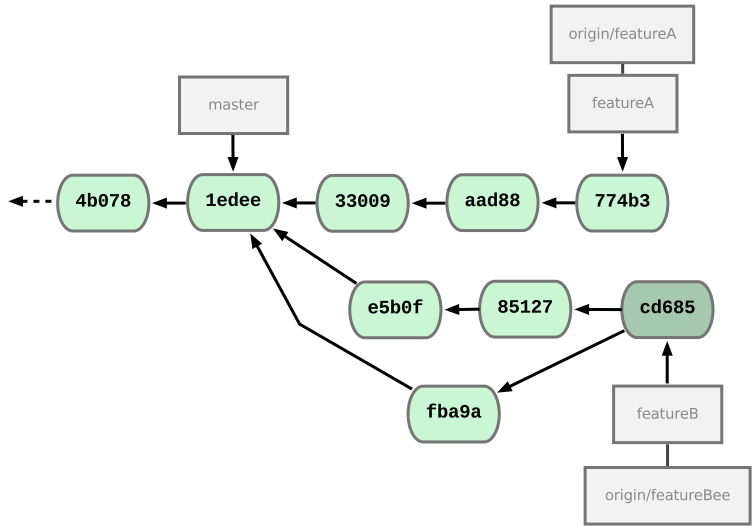
```
git flow feature
git flow release
```

- Unterstützung in Atlassian SourceTree [7]
- Eclipse EGit Wishlist Bug #348610

BRANCHES, TAGS, MERGE UND REBASE

MERGING

- Content-Tracking statt Patch-Serie
  (wie z.B. bei SVN)

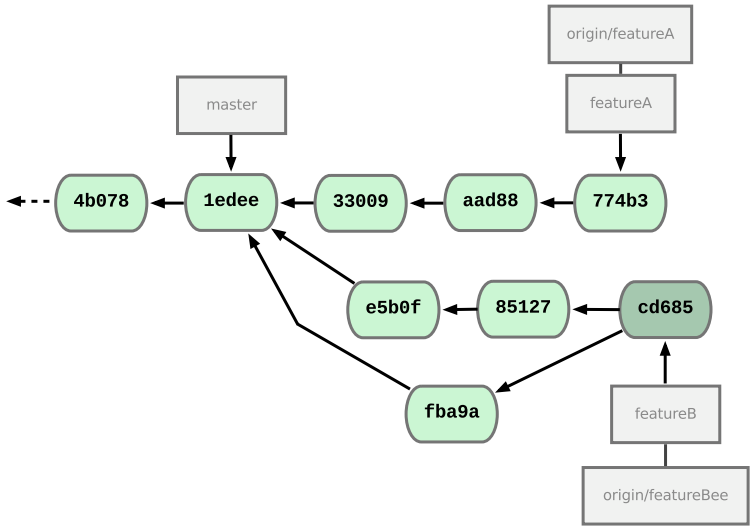


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

MERGING

- Content-Tracking statt Patch-Serie (wie z.B. bei SVN)
- es existieren mehrere Merge-Strategien (Standard recursive, mehr unter `git help merge`)

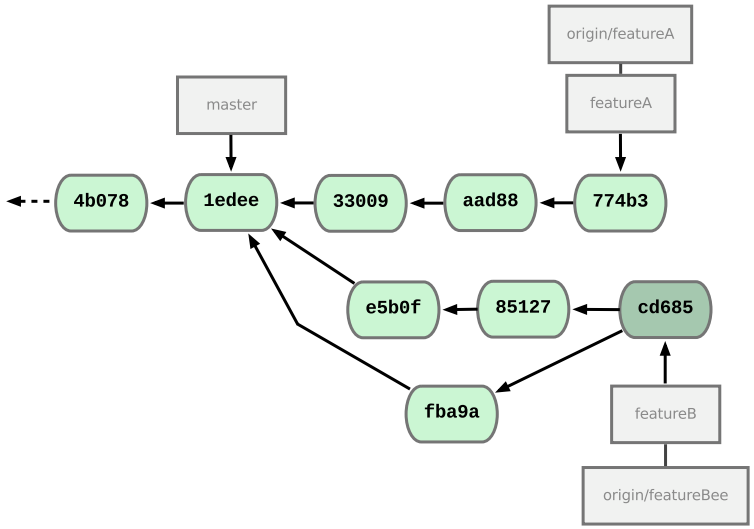


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

MERGING

- Content-Tracking statt Patch-Serie (wie z.B. bei SVN)
- es existieren mehrere Merge-Strategien (Standard recursive, mehr unter `git help merge`)
- Mergekonflikte müssen wie bei anderen VCS aufgelöst werden, dafür können Tools eingebunden werden

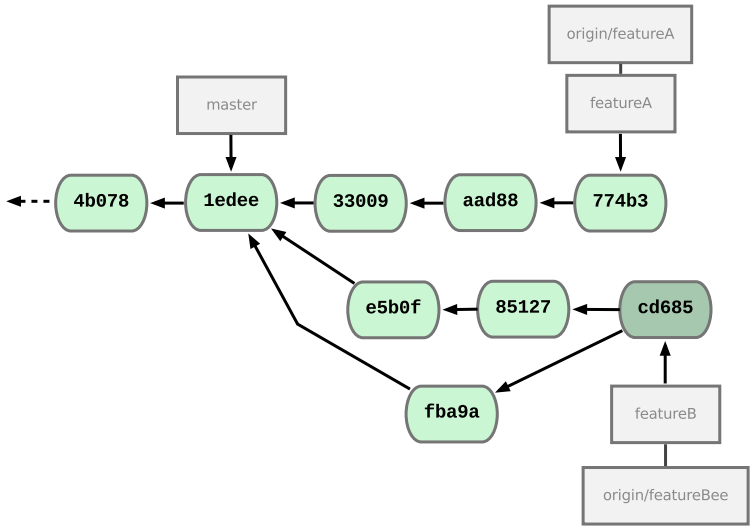


Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

MERGING

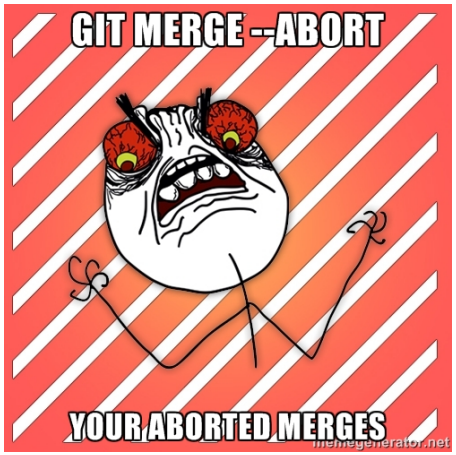
- Content-Tracking statt Patch-Serie (wie z.B. bei SVN)
- es existieren mehrere Merge-Strategien (Standard recursive, mehr unter `git help merge`)
- Mergekonflikte müssen wie bei anderen VCS aufgelöst werden, dafür können Tools eingebunden werden
- Kommando: `git merge`



Quelle: ProGit [4]

BRANCHES, TAGS, MERGE UND REBASE

MERGING



ACHTUNG:

Wenn ein Merge abgebrochen werden soll statt den Mergekonflikt zu lösen (z.B. mit `git reset --hard`), IMMER `git merge --abort` benutzen, da sonst der nachfolgende Commit automatisch ein Merge-Commit wird und die Änderungen aus dem ursprünglich für den Merge vorgesehenen Branch dann nicht enthalten sind.

BRANCHES, TAGS, MERGE UND REBASE

MERGE-TOOLS

- Eclipse mit EGit [8]

BRANCHES, TAGS, MERGE UND REBASE

MERGE-TOOLS

- Eclipse mit EGit [8]
- TortoiseMerge (mit TortoiseSVN oder TortoiseGit [9] installiert)

BRANCHES, TAGS, MERGE UND REBASE

MERGE-TOOLS

- Eclipse mit EGit [8]
- TortoiseMerge (mit TortoiseSVN oder TortoiseGit [9] installiert)
- vimdiff

BRANCHES, TAGS, MERGE UND REBASE

MERGE-TOOLS

- Eclipse mit EGit [8]
- TortoiseMerge (mit TortoiseSVN oder TortoiseGit [9] installiert)
- vimdiff
- kann konfiguriert (`git help config`, Suchbegriff **merge.tool**) und über `git mergetool` aufgerufen werden

BRANCHES, TAGS, MERGE UND REBASE

PATCH-WORKFLOW

Neben `git merge` kann auch mit Patches gearbeitet werden:

- `git format-patch` zum Erstellen,
- `git send-email` zum Versenden von Patch-Serien,
- `git am` (Apply Mail) zum Integrieren

Diese Vorgehensweise kommt noch recht häufig bei Free/Open Source Software Projekten zum Einsatz

BRANCHES, TAGS, MERGE UND REBASE

CHERRY-PICKING

- einzelne Commits aus anderem Branch in aktuellen Branch übernehmen

BRANCHES, TAGS, MERGE UND REBASE

CHERRY-PICKING

- einzelne Commits aus anderem Branch in aktuellen Branch übernehmen
- z.B. für Hotfixes

BRANCHES, TAGS, MERGE UND REBASE

CHERRY-PICKING

- einzelne Commits aus anderem Branch in aktuellen Branch übernehmen
- z.B. für Hotfixes
- `git cherry-pick <commits>` (`<commits>` kann mit Rev-Spezifikation definiert werden)

BRANCHES, TAGS, MERGE UND REBASE

REBASING

- History eines (lokalen) Branches umpflanzen

BRANCHES, TAGS, MERGE UND REBASE

REBASING

- History eines (lokalen) Branches umpflanzen
- `git rebase <branchname>`

BRANCHES, TAGS, MERGE UND REBASE

REBASING

- History eines (lokalen) Branches umpflanzen
- `git rebase <branchname>`
- ACHTUNG: rebasing von bereits publizierten Branches ist BÖSE

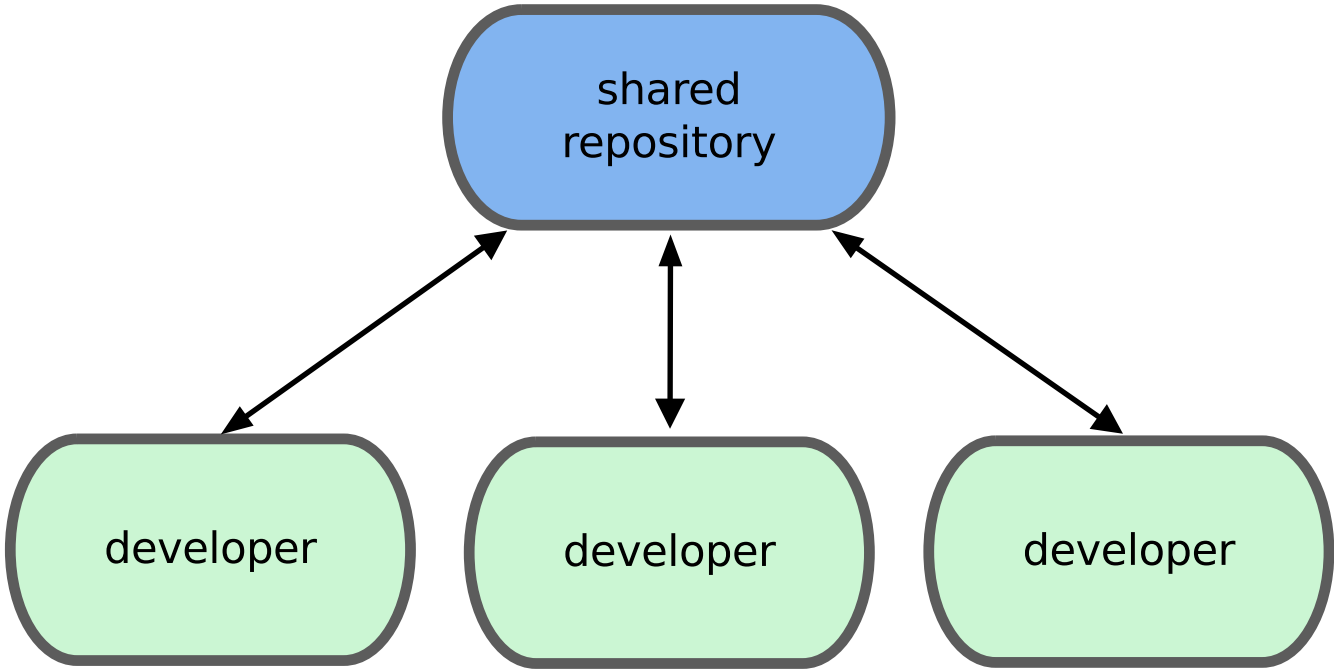
REPOSITORY-ORGANISATION

gebräuchliche Repository-Strukturen
Interaktion mit Repositories

REPOSITORY-ORGANISATION

GEBRÄUCHLICHE REPOSITORY-STRUKTUREN

zentrales Repository mit Entwickler-Clones (wie SVN)

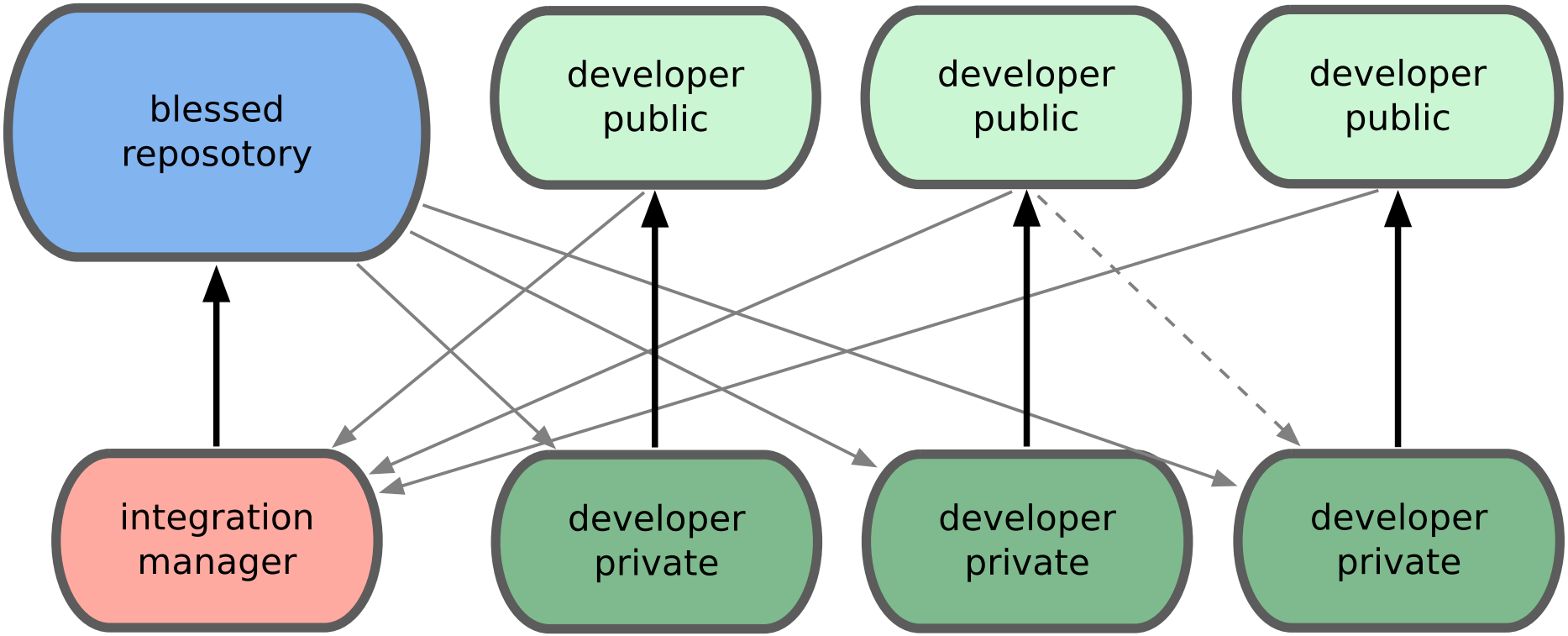


Quelle: ProGit [4]

REPOSITORY-ORGANISATION

GEBRÄUCHLICHE REPOSITORY-STRUKTUREN

mehrere zentrale Repositories (z.B. für QA, Entwickler, Kunden, ...)

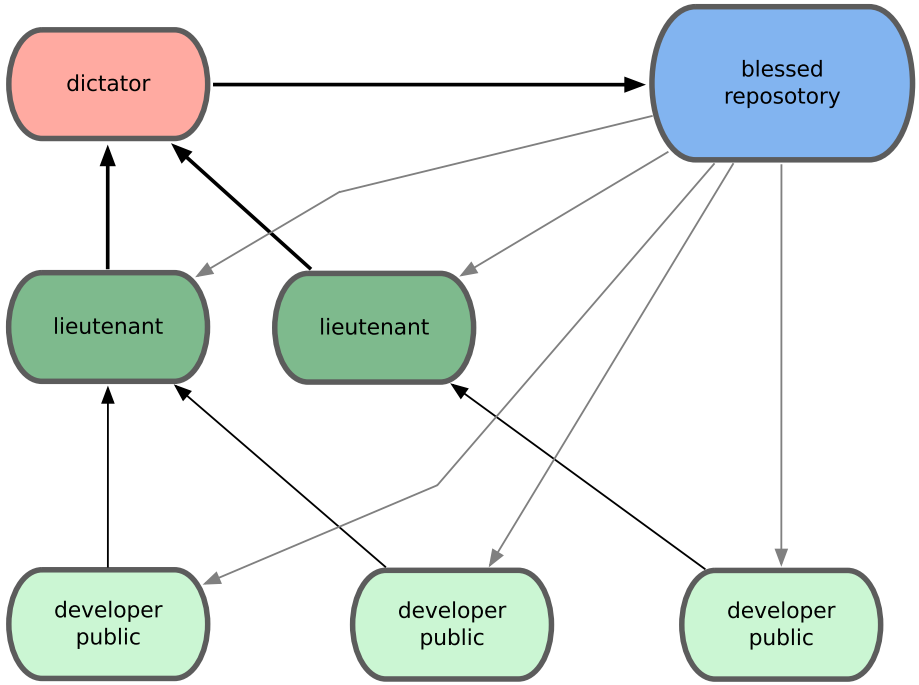


Quelle: ProGit [4]

REPOSITORY-ORGANISATION

GEBRÄUCHLICHE REPOSITORY-STRUKTUREN

mehrstufige Repositories (Dictator, Lieutenant)



Quelle: ProGit [4]

REPOSITORY-ORGANISATION
INTERAKTION MIT REPOSITORIES

- Klone erstellen

```
git clone <url>
```

REPOSITORY-ORGANISATION
INTERAKTION MIT REPOSITORIES

- Klone erstellen

  ```
  git clone <url>
  ```
- Umgang mit Remotes

REPOSITORY-ORGANISATION

INTERAKTION MIT REPOSITORIES

- Klone erstellen

  `git clone <url>`
- Umgang mit Remotes
  - hinzufügen

    `git remote add`

REPOSITORY-ORGANISATION

INTERAKTION MIT REPOSITORIES

- Klone erstellen

  `git clone <url>`
- Umgang mit Remotes
  - hinzufügen

    `git remote add`
  - löschen

    `git remote rm`

REPOSITORY-ORGANISATION

INTERAKTION MIT REPOSITORIES

- Klone erstellen

  `git clone <url>`
- Umgang mit Remotes
  - hinzufügen

    `git remote add`
  - löschen

    `git remote rm`
- Branches von remote abrufen

  `git fetch <remote> <branch>`

REPOSITORY-ORGANISATION

INTERAKTION MIT REPOSITORIES

- Klone erstellen

  `git clone <url>`
- Umgang mit Remotes
  - hinzufügen

    `git remote add`
  - löschen

    `git remote rm`
- Branches von remote abrufen

  `git fetch <remote> <branch>`
- Branch abrufen und mit HEAD mergen

  `git pull`

REPOSITORY-ORGANISATION

INTERAKTION MIT REPOSITORIES

- Klone erstellen

  `git clone <url>`
- Umgang mit Remotes
  - hinzufügen

    `git remote add`
  - löschen

    `git remote rm`
- Branches von remote abrufen

  `git fetch <remote> <branch>`
- Branch abrufen und mit HEAD mergen

  `git pull`
- lokale Commits/Tags publizieren

  `git push`

HOOKS

HOOKS

- Skripte in .git/hooks für Eingriff in verschiedenen Phasen

HOOKS

- Skripte in .git/hooks für Eingriff in verschiedenen Phasen
- lokal für jeweiliges Repository

HOOKS

- Skripte in .git/hooks für Eingriff in verschiedenen Phasen
- lokal für jeweiliges Repository
- werden bei `git clone` nicht kopiert

HOOKS

- Skripte in .git/hooks für Eingriff in verschiedenen Phasen
- lokal für jeweiliges Repository
- werden bei `git clone` nicht kopiert
- `git help hooks`

WEITERFÜHRENDES

WEITERFÜHRENDES

- Git-Webseite [10]
- Git Cheatsheet [11]
- changelog.com-Feed zu Git [12]
- nützliche Zusatzkommandos für die Shell `git-extras` [13]

WEITERFÜHRENDES

- Version Control with Git [14]
- Pro Git [4] (auch kostenlos online lesbar)
- Git: Dezentrale Versionsverwaltung im Team [15]

Teil II

GIT IN DER MMS

UNTERSTÜTZUNG FÜR PROJEKTE

UNTERSTÜTZUNG FÜR PROJEKTE

- Schulungen für Projekte
- Hilfe beim Projektsetup
- Know-How zu Git-Clients (CLI und Atlassian SourceTree)
- Beratung zu Best-Practice
- Git-Community im Teamweb
  `http://teamweb.mms-at-work.de/x/cRFDCQ`

MMS-TOOLS

Atlassian Stash
Unterstützte Clients
MMS spezifische Einrichtung

MMS-TOOLS
ATLASSIAN STASH

- Git-Repository-Manager von Atlassian
- bietet gute Integration in andere Atlassian-Produkte
- Rechtekonzept auf Basis von Active-Directory-Gruppen

MMS-TOOLS
UNTERSTÜTZTE CLIENTS

„Offizielle“ Unterstützung gibt es für die Nutzung folgender Clients:

- Git for Windows / Kommandozeile
- Atlassian SourceTree [7] (Atlassian Id erforderlich)

- POST-Buffergröße hochsetzen:
  `git config --global http.postBuffer 524288000`
- Proxy-Konfiguration in `/.bashrc`, `/.zshrc` bzw. Windows-Umgebungsvariablen:

```
http_proxy=http://proxy.mms-dresden.de:8080/
https_proxy=http://proxy.mms-dresden.de:8080/
no_proxy=git.t-systems-mms.eu,.mms-at-work.de, ↵
     .mms-dresden.de
```

  nur nötig wenn andere Zugriffe über den Proxy per Kommandozeile gemacht werden sollen

EMPFOHLENE BRANCHING-STRUKTUREN

Basis
Development-Branching
Feature/User-Branching
Release-Branching

EMPFOHLENE BRANCHING-STRUKTUREN

BASIS

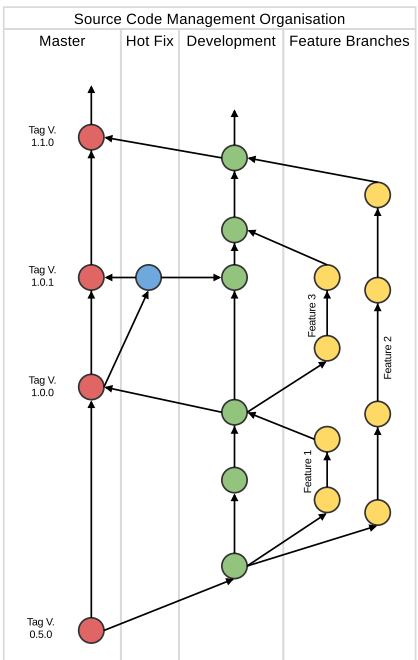
Branching-Modelle im Team-Web unter

`http://teamweb.mms-at-work.de/x/LoEfC`

EMPFOHLENE BRANCHING-STRUKTUREN

BASIS

Auswahl des passenden Branchingmodells

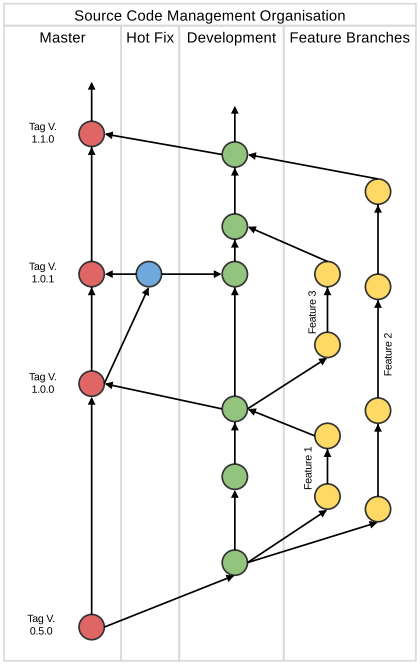


- mehrere unterstützte Branchingmodelle
- für unterschiedliche Projektgrößen und -anforderungen
- Basis für alle ist git-flow
- Branchingmodelle können bei Bedarf mit dem Projekt wachsen

EMPFOHLENE BRANCHING-STRUKTUREN

BASIS

Standardbranches



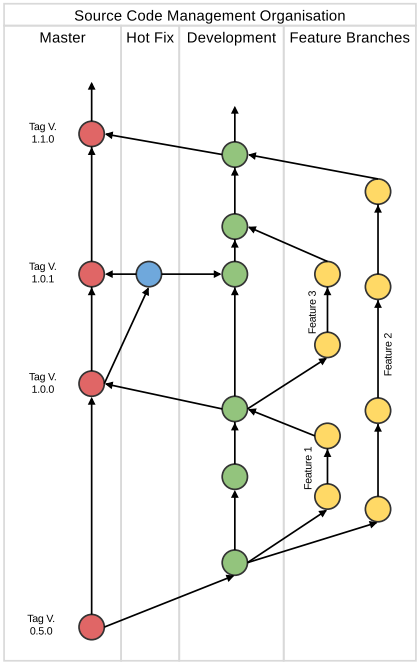
master

- Hauptzweig für Lieferungen
- normalerweise read-only
- nur qualitätsgesicherte Stände aus dem development-Branch

EMPFOHLENE BRANCHING-STRUKTUREN

BASIS

Standardbranches



master

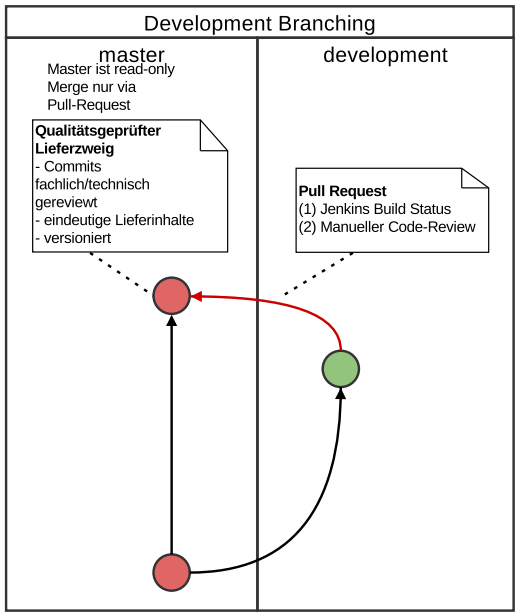
- Hauptzweig für Lieferungen
- normalerweise read-only
- nur qualitätsgesicherte Stände aus dem development-Branch

development

- Entwicklungszweig ähnlich trunk bei SVN
- entweder direkte Commits (sehr kleine Teams/Projekte)
- ...oder merge von Feature-/User-Branches

EMPFOHLENE BRANCHING-STRUKTUREN

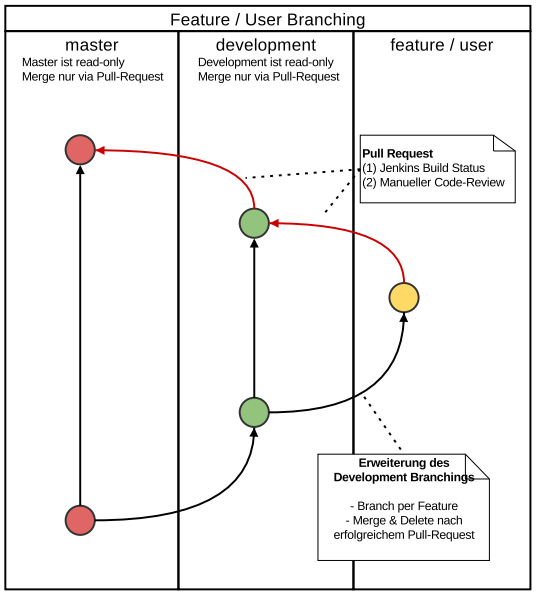
DEVELOPMENT-BRANCHING



- für sehr kleine Teams/Projekte
- mit klarem Scope und kleinen unabhängigen Features
- z.B. für erstes/einziges Release

EMPFOHLENE BRANCHING-STRUKTUREN

FEATURE/USER-BRANCHING

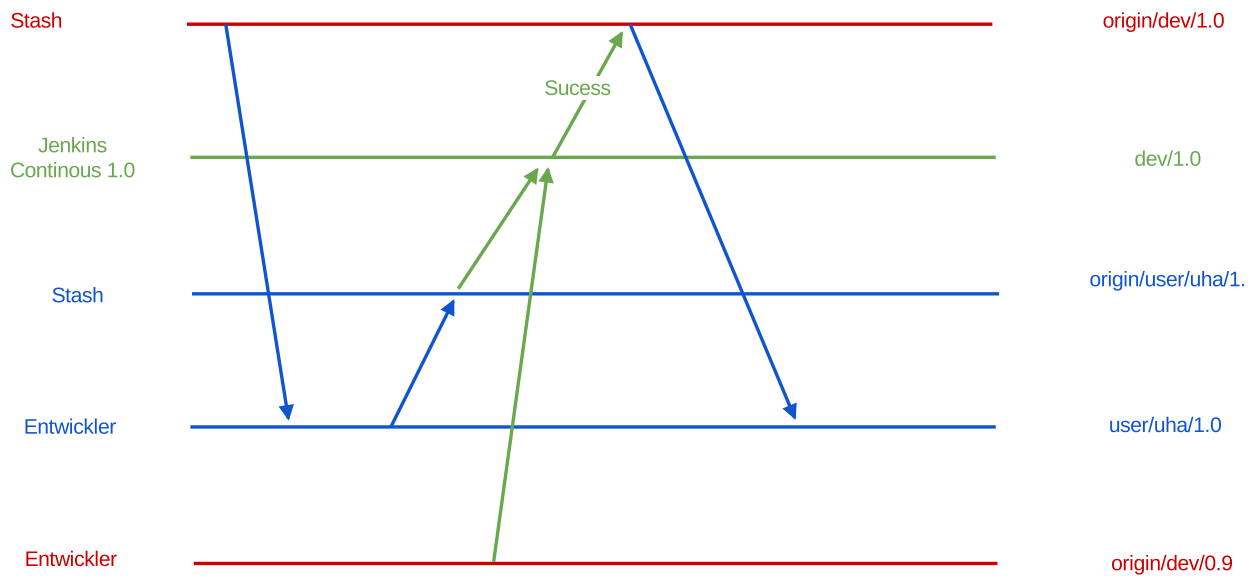


- größere Teams und/oder große Feature
- Feature-/User-Branches werden nach Qualitätssicherung in den **development**-Branch übernommen

EMPFOHLENE BRANCHING-STRUKTUREN

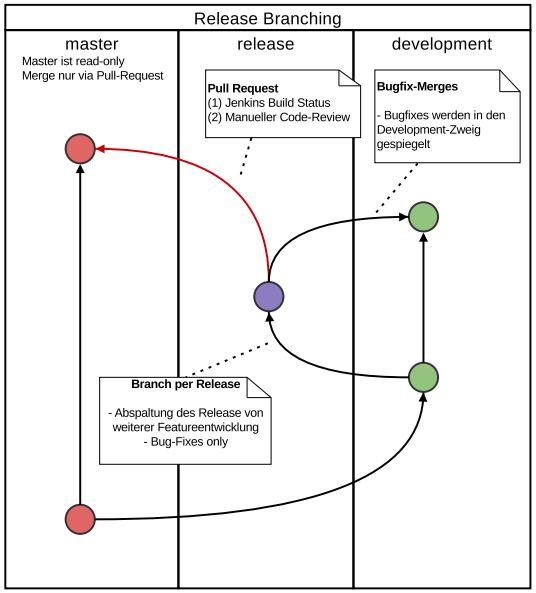
FEATURE/USER-BRANCHING

Variante: User-Branching



EMPFOHLENE BRANCHING-STRUKTUREN

RELEASE-BRANCHING



- Erweiterung für Development- und Feature-/User-Branching
- für unabhängige Entwicklung paralleler Releases

EINBINDUNG IN DIE TOOLKETTE

Integration Git-Client mit Stash
Integration Stash mit JIRA
Integration Git mit Jenkins

EINBINDUNG IN DIE TOOLKETTE

INTEGRATION GIT-CLIENT MIT STASH

- Git-Repository wird auf dem Client geklont (entweder direkt oder mit einem projektspezifischen Setup-Skript)
- Änderungen des Clients werden wieder zu Stash gepusht
- Commits können über Commit-Kommentare auch weitere Aktionen triggern
- Repository-Hooks werden bei Stash mit Plugins umgesetzt

EINBINDUNG IN DIE TOOLKETTE

INTEGRATION STASH MIT JIRA

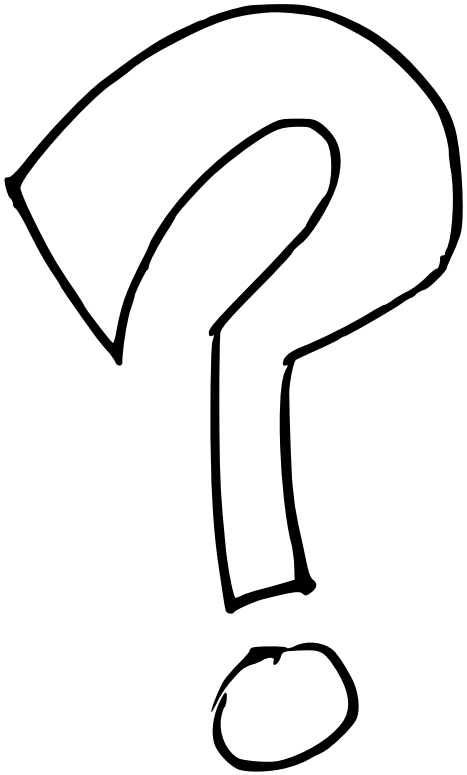
- Wenn ein Commit-Kommentar oder Branchname eine JIRA-Ticket-Id enthält, wird der Commit mit dem JIRA-Ticket verknüpft
- Aus JIRA heraus können, bei entsprechendem Projekt-Setup, Ticket-basierte Feature-Branches erstellt werden

EINBINDUNG IN DIE TOOLKETTE

INTEGRATION GIT MIT JENKINS

- Stash-Plugin benachrichtigt Jenkins
- Jenkins baut Projekt, führt Tests durch, etc.
- Jenkins benachrichtigt optional Stash
- Jenkins kann optional auch Kommentare in JIRA hinterlassen
- **Hinweis:** funktioniert im MMS-LAN reibungsloser als im T-LAN

FRAGEN



Zeit für Fragen

KONTAKT

Jan Dittberner

E-Mail: jan.dittberner@t-systems.com

Telefon: +49-351-2802-2737

Twitter/Identi.ca: @jandd

LITERATUR

[1] Cygwin. URL: `http://www.cygwin.com/`.

[2] msysgit – Git for Windows. URL: `http://msysgit.github.com/`.

[3] Mac OS-X Installer for Git. URL: `http://git-scm.com/download/mac`.

[4] Scott Chacon. Pro Git. Apress, 2009. ISBN: 978-1-4302-1833-3. URL: `http://git-scm.com/book/`.

[5] Vincent Driessen. A successful Git branching model. 2010. URL: `http://nvie.com/posts/a-successful-git-branching-model/`.

[6] Vincent Driessen. gitflow. URL: `https://github.com/nvie/gitflow`.

LITERATUR

[7] Atlassian SourceTree, A free Mercurial and Git client for Windows or Mac. URL: `http://atlassian.com/software/sourcetree/overview`.

[8] EGit – Eclipse Integration for Git. URL: `http://www.eclipse.org/egit/`.

[9] TortoiseGit. URL: `http://code.google.com/p/tortoisegit/`.

[10] Git project website. URL: `http://www.git-scm.com/`.

[11] Git Cheatsheet – Categorize Git's commands based on what they affect. URL: `http://ndpsoftware.com/git-cheatsheet.html`.

[12] Posts tagged Git on changelog.com. URL: `http://thechangelog.com/tagged/git/`.

LITERATUR

[13] **git-extras.** URL: `https://github.com/visionmedia/git-extras#readme`.

[14] Jon Loeliger and Matthew McCullough. **Version Control with Git, 2nd Edition.** O'Reilly Media, 2012. ISBN: 978-1-4493-1638-9. URL: `http://shop.oreilly.com/product/0636920022862.do`.

[15] René Preißel and Bjørn Stachmann. **Git: Dezentrale Versionverwaltung im Team - Grundlagen und Workflows.** dpunkt.verlag GmbH, 2013. ISBN: 978-3-8649-0130-0. URL: `http://www.dpunkt.de/buecher/4518/git.html`.