VERSIONEN DEZENTRAL VERWALTEN MIT GIT

Dipl.-Ing. (FH) Eugen Richter

GIT: THEMEN

- Warum Versionierung?
- Geschichte der Versionsverwaltung (allgemein)
- Geschichte von Git (im besonderen)
- Übersicht über Versionierungsstrategien
- Git auf der Console (Kommandozeile)
- Git mit SourceTree (als Beispiel für eine graphische Oberfläche)

WARUM VERSIONIERUNG?

- Datensicherung
-
-
-

WARUM VERSIONIERUNG?

- Datensicherung
- Älterer Zustand
-
-

WARUM VERSIONIERUNG?

- Datensicherung
- Älterer Zustand
- Parallele-Arbeit an mehreren Versionen
-

WARUM VERSIONIERUNG?

- Datensicherung
- Älterer Zustand
- Parallele-Arbeit an mehreren Versionen
- Parallele-Arbeit mit mehreren Personen

GESCHICHTE DER VERSIONIERUNG

- Zeitstempel-Ordner
-
-
-

GESCHICHTE DER VERSIONIERUNG

- Zeitstempel-Ordner
- Dateiversionierung
-
-

GESCHICHTE DER VERSIONIERUNG

- Zeitstempel-Ordner
- Dateiversionierung
- Zentral
-

GESCHICHTE DER VERSIONIERUNG

- Zeitstempel-Ordner
- Dateiversionierung
- Zentral
- Verteilt

GESCHICHTE VON GIT

- 2005 von Linus Torwalds initialisiert
-
-
-
-
-

GESCHICHTE VON GIT

- 2005 von Linus Torwalds initialisiert
- Erste Version in wenigen Tagen
-
-
-
-

GESCHICHTE VON GIT

- 2005 von Linus Torwalds initialisiert
- Erste Version in wenigen Tagen
- Zur Verwaltung von Linux Kernel (sehr verteilte Entwicklung)
-
-
-

GESCHICHTE VON GIT

- 2005 von Linus Torwalds initialisiert
- Erste Version in wenigen Tagen
- Zur Verwaltung von Linux Kernel (sehr verteilte Entwicklung)
- Sehr hohe Effizienz
-
-

GESCHICHTE VON GIT

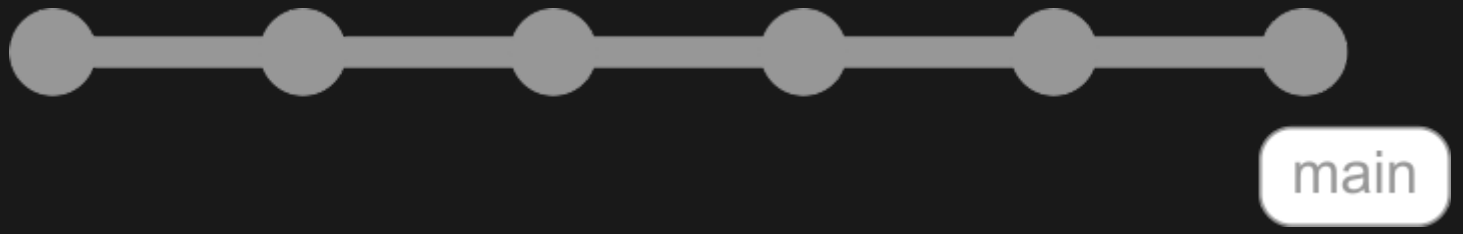
- 2005 von Linus Torwalds initialisiert
- Erste Version in wenigen Tagen
- Zur Verwaltung von Linux Kernel (sehr verteilte Entwicklung)
- Sehr hohe Effizienz
- Sehr hohe Sicherheit
-

GESCHICHTE VON GIT

- 2005 von Linus Torwalds initialisiert
- Erste Version in wenigen Tagen
- Zur Verwaltung von Linux Kernel (sehr verteilte Entwicklung)
- Sehr hohe Effizienz
- Sehr hohe Sicherheit
- Wegwerf-Zweige

VERSIONSSTRATEGIEN

LINEARE ENTWICKLUNG



EIN BRANCH - PRO

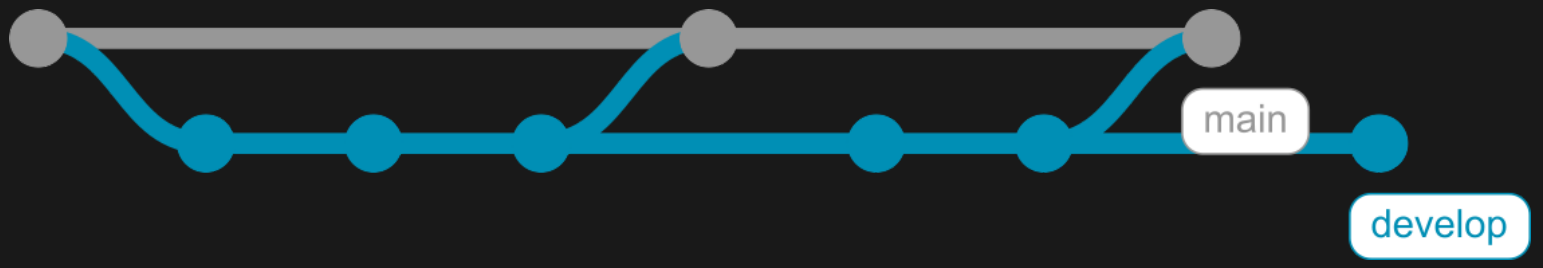
- Sehr einfache Benutzung
- Kein Merge zwischen unterschiedlichen Zweigen notwendig
- Sehr gut für den Einstieg in die Versionsverwaltung geeignet
- Sehr gut für Dokument-Versionierung (Bücher, Artikel, Manuskripte usw.)

EIN BRANCH - CONTRA

- Schwer zu handhaben, wenn mehr als nur ein Entwickler beteilig ist, da während des Release-Tests keine Weiterentwicklung für nächste Version möglich ist.
- Hotfixes einer Version sind sehr schwer zu realisieren, da eventuell bereits unvollständige Features für neue Version da sind.

MAIN - DEVELOP

STABLILER UND ENTWICKLUNGSZWEIG



MAIN - DEVLOP - PRO

- Bietet besseren Überblick über ausgelieferte / veröffentlichte Projektstände und belässt die Flexibilität bei der täglichen Arbeit.
- Schneller Zugriff auf benannte Stände, da diese nur im Master-Zweig vertreten sind (ohne Entwicklungsbalast).

MAIN - DEVELOP - CONTRA

- Schwer zu handhaben, wenn mehr als nur ein Entwickler beteilig ist, da während des Release-Tests keine Weiterentwicklung für nächste Version möglich ist.
- Hotfixes einer Version sind sehr schwer zu realisieren, da eventuell bereits unvollständige Features für neue Version da sind.

GIT-FLOW

MAIN, DEVELOP, FEATURE, RELEASE, HOTFIX

GIT FLOW – PRO

- Arbeiten im Team ohne Beeinträchtigungen möglich, da Aufgaben in eigenen Zweigen erledigt werden.
- Saubere Implementierung der Hotfixes für Release-Versionen ohne Beeinträchtigung der Entwicklung möglich.
- Paralelles Weiterentwicklen der nächsten Version und vorbereiten (testen) der aktuellen durch Release-Zweige möglich.
- Auslieferung von Hotfixes in sehr kurzer Zeit möglich, da keine Rücksicht auf den Entwicklungsstand genommen werden muss.

GIT FLOW – CONTRA

WEITERE STRATEGIEN

- Forking
-
-

WEITERE STRATEGIEN

- Forking
- Pull Request
-

WEITERE STRATEGIEN

- Forking
- Pull Request
- GitHub Flow

WORKSHOP