Dienstag: Ressourcen

3 Ressourcen
- Einführung
- Korpora

Ressourcen

Ressourcen

Ressourcen

Korpora

- Annotierte und nicht-annotierte Textmengen
- BNC, ANC, Europarl, FrameNet, Salsa, Negra, Wikipedia, ...
- Kein einheitliches Format!

Tools

- Programme für die linguistische (und andere) Verarbeitung
- POS-Tagger, Parser, Sentence Splitter, Machine Learning Toolkits, ...
- Jedes Programm verhält sich anders!

Dokumentation I

https://wiki.cl.uni-heidelberg.de/foswiki/bin/view/Main/Resources/WebHome

Webseite

- Login mit ICL-Account
- Wikiseite enthält Index und detaillierte Beschreibung der verfügbaren Ressourcen

Dokumentation II

https://wiki.cl.uni-heidelberg.de/foswiki/bin/view/Main/Resources/WebHome

Index

- Oben: Kategorien
- Unten: Alphabetische Liste der Ressourcen nach Kategorien mit
  - Links zur Wikiseite (enthält Link zur Homepage und Dokumentation)
  - Kurzbeschreibung
  - Verzeichnispfad

Dokumentation III

Weiterführende Dokumentation zu einzelnen Tools

- README
- man-pages
- doc-Verzeichnis
- ...

Vertraulichkeitsvereinbarung I

- Non Disclosure Agreement (NDA)
- Für manche Ressourcen ist eine Unterschrift nötig, bevor damit gearbeitet werden kann.
- Damit übernehmen Sie die Verantwortung dafür, dass die Ressourcen nicht über Sie an unberechtigte Dritte gelangen.
- Die betroffenen Ressourcen sind mit einem kleinen Schloss-Symbol gekennzeichnet.
- `https://wiki.cl.uni-heidelberg.de/foswiki/bin/view/Main/Resources/NDA`

Vertraulichkeitsvereinbarung II

Howto

1. NDA bei Gruppe Technik abgeben (`http://www.cl.uni-heidelberg.de/gruppetechnik/`)
2. In die Gruppe **`resuser`** aufgenommen werden

Ressourcen

Kategorien I

Statistics/ML Tools für Statistik und Machine Learning

Processors Software für *einen* Verarbeitungsschritt: Parser, Tagger, ...

Preprocessors Software zur Vorverarbeitung (z.B. HTML-Extraktion)

Platforms Toolkits/Frameworks, die verschiedene Verarbeitungsschritte beinhalten (z.B. NLTK)

Ontologies Ontologien und WordNet

Kategorien II

Lingware	Linguistische Software: Grammatiken, Morphologien, ...
Corpora	Alle Korpora
APIs	*Application Programming Interfaces* – Interfaces, um aus Programmen auf Ressourcen zuzugreifen
Annotation	Annotationswerkzeuge

Unterkategorien für Korpora

- Manche Kategorien haben Unterkategorien

Corpora and Data

- Monolingual Corpora
- Multilingual Corpora
- Speech
- Viewers

Ressourcen

Setup

- Einstellungen für manche Tools
- Oft: Umgebungsvariablen (`$PATH`, `$LIBRARY_PATH`, ...)
- Datei `setup` in jedem Ressourcen-Verzeichnis
- Aktivierung mittels `source`

source

- Eingebaut in die Shell
- Erwartet als Argument eine Datei, in der Shell-Kommandos stehen
- Kommandos werden der Reihe nach ausgeführt *ohne eine Subshell zu starten*
- Notwendig, damit Änderungen an Umgebungsvariablen in der Shell vorgenommen werden können
- Beispiel:

```
:~$ source /resources/path/to/resource/setup
```

Troubleshooting

Was tun wenn es nicht klappt?

- *Don't Panic*!
- Häufigster Fehler: **`setup`**-Skript nicht ausgeführt
- Viele mögliche Ursachen
- Dokumentation lesen und nachvollziehen
- Kontakt: `resources@cl.uni-heidelberg.de` (Englisch)

Fehlerbeschreibungen

Was sollte eine sinnvolle Fehlerbeschreibung enthalten?

1. *Alles, was man braucht, um das Problem zu reproduzieren*
   1. Vor allem: die Fehlermeldung!
   2. Der Code, der den Fehler erzeugt
   3. Auf welchem Rechner passiert das ganze?
      Betriebssystem, Username, Verzeichnis, ...
2. Was haben Sie bereits versucht? Was haben Sie als letztes geändert?
3. Benutzen Sie einen *sinnvollen Betreff*
4. `http://www.cl.uni-heidelberg.de/computerpool/technikinfo/`

Ressourcen

Ressourcen

Korpora

- Entweder mit Annotationen versehen oder als reiner Text.
- Meistens in Abschnitte unterteilt (z.B. einzelne Dokumente, Sitzungen, Gespräche, Quellen).
- Verschiedene Annotationsstile sind gängig, je nachdem, was annotiert wird.
- Es hilft im Kopf zu behalten, wie ein Korpus erstellt wurde!

Unannotierte Korpora

- Daten kaum vorverarbeitet.
- Nützliche Information in der Herkunft eines Dokuments:
  - Datum
  - Ort
  - Autor
  - ...

Annotierte Korpora I

- Zwei Annotationsweisen:

Inline Annotation

- Annotation direkt ins Dokument eingefügt:
  `the <noun>dog</noun> barks.`

Stand-Off Annotation

- Annotation getrennt vom Dokument:
- Zeichen- oder Wortpositionen verweisen auf die Stelle im Text
  `<noun start="4" end="7" />`

Inline Annotation I

- Verändert das Dokument

Beispiel

- `the dog barks`
- + POS Tags: `<d>the</d> <n>dog</n> <v>barks</v>`
- + Chunks:
  `<np><d>the</d> <n>dog</n></np>`
  `<vp><v>barks</v></vp>`

Inline Annotation II

Nachteile

- Je mehr Annotation, umso schwieriger, die richtige Stelle zu finden.
- Nicht möglich, z.B. zwei verschiedene Parsebäume darzustellen
- Überschneidungen/Überlappungen sind schwer zu modellieren (`<a>just <a>an</a> example</a>`)

Stand-Off Annotation

- Originaldokument bleibt unverändert.
- Beliebige Annotation möglich.
- Annotatierte Daten schwer lesbar (für menschliche Leser).

Unbedingt beachten:

- Bezieht sich die Annotation auf das letzte Zeichen innerhalb einer Spanne oder auf das erste Zeichen außerhalb der Spanne?
- Bezieht sich die Nummerierung auf Bytes (unicode!), Zeichen oder Wörter?

Stand-Off Annotation

- Originaldokument bleibt unverändert.
- Beliebige Annotation möglich.
- Annotatierte Daten schwer lesbar (für menschliche Leser).

Unbedingt beachten:

- Bezieht sich die Annotation auf das letzte Zeichen innerhalb einer Spanne oder auf das erste Zeichen außerhalb der Spanne?
- Bezieht sich die Nummerierung auf Bytes (unicode!), Zeichen oder Wörter?

Gemischte Annotation

- Es wird kompliziert, wenn beide Annotationsstile vermischt werden.
- Beeinflusst Inline Annotation die Zeichenpositionen?
  - ja: Wenn Inline Annotation hinzugefügt wird müssen Zeichenpositionen neu berechnet werden!
  - nein: Man kann das annotierte Dokument parsen, um die ursprünglichen Positionen zu extrahieren!

- Tipp: Bei gemischter Annotation alle Inline Annotationen in Stand-Off Annotation konvertieren.

Gemischte Annotation

- Es wird kompliziert, wenn beide Annotationsstile vermischt werden.
- Beeinflusst Inline Annotation die Zeichenpositionen?
  - ja: Wenn Inline Annotation hinzugefügt wird müssen Zeichenpositionen neu berechnet werden!
  - nein: Man kann das annotierte Dokument parsen, um die ursprünglichen Positionen zu extrahieren!

- Tipp: Bei gemischter Annotation alle Inline Annotationen in Stand-Off Annotation konvertieren.

Ressourcen

Erstellung von Korpora

- (Die meisten) Texte werden nicht zum Zwecke der Korpuserstellung geschrieben/gesprochen.
- Ausnahme: Korpora für gesprochene Sprache häufig von bezahlten Sprechern in kontrollierten Szenarien erstellt (=€€€!)
- Textkorpora oft aus verschiedene Quellen zusammengestellt.

Quellen

- Bücher, Artikel, sonstige Veröffentlichungen
- Webseiten
- Tonaufnahmen
- Zeitungs- und Nachrichtentexte

Konvertierung von Formaten

- Dateien in einem Ordner genügen nicht.
- Das Korpus in einem nutzbaren Textformat sein.
- Originalformat muss häufig konvertiert werden.

Konvertierung	Tool
PDF $\rightarrow$ TXT	**`pdftotext`** aber: Bindestriche, Text in Spalten
HTML $\rightarrow$ TXT	**`python`** , **`perl`** , **`bash`** , ...

Tabelle: Konvertierungswerkzeuge

Aufbereitung

- Mehrere Schritte, je nach Bedarf
- 1: Sentence splitting
- 2: Tokenisierung
- 3: POS-Annotation, Lemmatisierung
- 4: Annotation von Eigennamen
- 5: Annotation der Satzstruktur
- n: ...
- = Pipeline

Aufbereitung

- Mehrere Schritte, je nach Bedarf
- 1: Sentence splitting
- 2: Tokenisierung
- 3: POS-Annotation, Lemmatisierung
- 4: Annotation von Eigennamen
- 5: Annotation der Satzstruktur
- n: ...
- = Pipeline

Aufbereitung

- Mehrere Schritte, je nach Bedarf
- 1: Sentence splitting
- 2: Tokenisierung
- 3: POS-Annotation, Lemmatisierung
- 4: Annotation von Eigennamen
- 5: Annotation der Satzstruktur
- n: ...
- = Pipeline

Aufbereitung

- Mehrere Schritte, je nach Bedarf
- 1: Sentence splitting
- 2: Tokenisierung
- 3: POS-Annotation, Lemmatisierung
- 4: Annotation von Eigennamen
- 5: Annotation der Satzstruktur
- n: ...
- = Pipeline

Aufbereitung

- Mehrere Schritte, je nach Bedarf
- 1: Sentence splitting
- 2: Tokenisierung
- 3: POS-Annotation, Lemmatisierung
- 4: Annotation von Eigennamen
- 5: Annotation der Satzstruktur
- n: ...
- = Pipeline

Aufbereitung

- Mehrere Schritte, je nach Bedarf
- 1: Sentence splitting
- 2: Tokenisierung
- 3: POS-Annotation, Lemmatisierung
- 4: Annotation von Eigennamen
- 5: Annotation der Satzstruktur
- n: ...
- = Pipeline

Aufbereitung

- Mehrere Schritte, je nach Bedarf
- 1: Sentence splitting
- 2: Tokenisierung
- 3: POS-Annotation, Lemmatisierung
- 4: Annotation von Eigennamen
- 5: Annotation der Satzstruktur
- n: ...
- = Pipeline

Aufbereitung

- Mehrere Schritte, je nach Bedarf
- 1: Sentence splitting
- 2: Tokenisierung
- 3: POS-Annotation, Lemmatisierung
- 4: Annotation von Eigennamen
- 5: Annotation der Satzstruktur
- n: ...
- = Pipeline

Pipeline-Werkzeuge

- Bestimmte Frameworks können helfen:
  - Datenstrukturen
  - Interfaces
  - Parallelisierung

Beispiel

- UIMA: `http://incubator.apache.org/uima/`
- OpenNLP: `http://opennlp.sourceforge.net/`
- GATE: `http://gate.ac.uk/`
- Heart of Gold: `http://heartofgold.dfki.de/`

Pipeline-Werkzeuge

- Bestimmte Frameworks können helfen:
  - Datenstrukturen
  - Interfaces
  - Parallelisierung

Beispiel

- UIMA: `http://incubator.apache.org/uima/`
- OpenNLP: `http://opennlp.sourceforge.net/`
- GATE: `http://gate.ac.uk/`
- Heart of Gold: `http://heartofgold.dfki.de/`

Ressourcen

Unannotierte Korpora

- Projekt Gutenberg: Sammlung von Texten mit abgelaufenem Urheberrecht.
- WAC: gecrawlte Webseiten von verschiedenen Domains (.de, .uk, .it, ...)
  Teilweise aufbereitete Korpora vorhanden (PukWAC)
- Web1t: 5-gramme aus dem Netz, erstellt von Google (1 TB!)
- Wortschatz: 3 Millionen deutsche Sätze

Unannotierte Korpora

- Projekt Gutenberg: Sammlung von Texten mit abgelaufenem Urheberrecht.
- WAC: gecrawlte Webseiten von verschiedenen Domains (.de, .uk, .it, ...)
  Teilweise aufbereitete Korpora vorhanden (PukWAC)
- Web1t: 5-gramme aus dem Netz, erstellt von Google (1 TB!)
- Wortschatz: 3 Millionen deutsche Sätze

Unannotierte Korpora

- Projekt Gutenberg: Sammlung von Texten mit abgelaufenem Urheberrecht.
- WAC: gecrawlte Webseiten von verschiedenen Domains (.de, .uk, .it, ...)
  Teilweise aufbereitete Korpora vorhanden (PukWAC)
- Web1t: 5-gramme aus dem Netz, erstellt von Google (1 TB!)
- Wortschatz: 3 Millionen deutsche Sätze

Unannotierte Korpora

- Projekt Gutenberg: Sammlung von Texten mit abgelaufenem Urheberrecht.
- WAC: gecrawlte Webseiten von verschiedenen Domains (.de, .uk, .it, ...)
  Teilweise aufbereitete Korpora vorhanden (PukWAC)
- Web1t: 5-gramme aus dem Netz, erstellt von Google (1 TB!)
- Wortschatz: 3 Millionen deutsche Sätze

Annotatierte Korpora

- BNC: Standardkorpus, „*balanced*", manuell annotiert mit POS und Lemma.
- Penn TreeBank: Nachrichtentexte mit manueller syntaktischer Annotation
- Negra: Deutsche Zeitungstexte, manuell annotiert mit POS (keine Lemmata) und syntaktischer Struktur.
- FrameNet: Englische Zeitungstexte, annotiert mit semantische Rollen nach dem FrameNet Paradigma.
- TimeBank: Nachrichtentext mit Events, Temporalausdrücken und temporalen Relationen zwischen Events.

Annotatierte Korpora

- BNC: Standardkorpus, „*balanced*", manuell annotiert mit POS und Lemma.
- Penn TreeBank: Nachrichtentexte mit manueller syntaktischer Annotation
- Negra: Deutsche Zeitungstexte, manuell annotiert mit POS (keine Lemmata) und syntaktischer Struktur.
- FrameNet: Englische Zeitungstexte, annotiert mit semantische Rollen nach dem FrameNet Paradigma.
- TimeBank: Nachrichtentext mit Events, Temporalausdrücken und temporalen Relationen zwischen Events.

Annotatierte Korpora

- BNC: Standardkorpus, „*balanced*", manuell annotiert mit POS und Lemma.
- Penn TreeBank: Nachrichtentexte mit manueller syntaktischer Annotation
- Negra: Deutsche Zeitungstexte, manuell annotiert mit POS (keine Lemmata) und syntaktischer Struktur.
- FrameNet: Englische Zeitungstexte, annotiert mit semantische Rollen nach dem FrameNet Paradigma.
- TimeBank: Nachrichtentext mit Events, Temporalausdrücken und temporalen Relationen zwischen Events.

Annotatierte Korpora

- BNC: Standardkorpus, „*balanced*", manuell annotiert mit POS und Lemma.
- Penn TreeBank: Nachrichtentexte mit manueller syntaktischer Annotation
- Negra: Deutsche Zeitungstexte, manuell annotiert mit POS (keine Lemmata) und syntaktischer Struktur.
- FrameNet: Englische Zeitungstexte, annotiert mit semantische Rollen nach dem FrameNet Paradigma.
- TimeBank: Nachrichtentext mit Events, Temporalausdrücken und temporalen Relationen zwischen Events.

Annotatierte Korpora

- BNC: Standardkorpus, „*balanced*", manuell annotiert mit POS und Lemma.
- Penn TreeBank: Nachrichtentexte mit manueller syntaktischer Annotation
- Negra: Deutsche Zeitungstexte, manuell annotiert mit POS (keine Lemmata) und syntaktischer Struktur.
- FrameNet: Englische Zeitungstexte, annotiert mit semantische Rollen nach dem FrameNet Paradigma.
- TimeBank: Nachrichtentext mit Events, Temporalausdrücken und temporalen Relationen zwischen Events.

Multilinguale Korpora

- Einige Korpora beinhalten Text in mehreren Sprachen.
- Multilinguale Quellen:
  - Wikipedia
  - Internationale Organisationen (EU, UN)
  - Übersetzte Texte (z.B. Nachrichtenagenturen, Projekt Gutenberg, mehrsprachige Webseiten)
- Parallel: Derselbe Text in verschiedenen Sprachen.
  - Europarl, OpenSubtitles, etc.
- Vergleichbar: Texte zum selben Thema in verschiedenen Sprachen
  - Reuters, WikiXML, ISI

Multilinguale Korpora

- Einige Korpora beinhalten Text in mehreren Sprachen.
- Multilinguale Quellen:
  - Wikipedia
  - Internationale Organisationen (EU, UN)
  - Übersetzte Texte (z.B. Nachrichtenagenturen, Projekt Gutenberg, mehrsprachige Webseiten)
- Parallel: Derselbe Text in verschiedenen Sprachen.
  - Europarl, OpenSubtitles, etc.
- Vergleichbar: Texte zum selben Thema in verschiedenen Sprachen
  - Reuters, WikiXML, ISI

Alignierungen

- Eine Alignierung (*alignment*) verbindet korrespondierende Passagen in verschiedenen Dokumenten.
- Parallele Korpora können auf verschiedenen Ebenen aligniert werden
- Dokument-, Absatz-, Satz-, Phrasen- und Wortalignierung
- Alignierung normalerweise implizit dargestellt durch IDs von Textpassagen.

Übung 4

Dienstag: Markup/Datenformate

4 Markup/Datenformate
- XML
- JSON

Markup/Datenformate

Motivation

- Viele Korpora enthalten Informationen über den reinen Text hinaus:
  - Metadaten
  - Annotationen
  - Dokumentgrenzen
- Annotationen oder Metadaten sind Eigenschaften von Texten
- Standard für Darstellung der Eigenschaften von Daten: XML
- Vorteil: Standard genau definiert, Bibliotheken für viele Programmiersprachen vorhanden

Markup/Datenformate

eXtensible Markup Language I

- XML ermöglicht die Speicherung und den Austausch strukturierter Daten über Textdateien
- Schreiben und Lesen mit normalem Text-Editor (emacs, vi, kate)
- XML-Parser erkennen die Struktur von XML-formatierten Daten
- Heutzutage sind viele Daten XML-formatiert (z.B. ODF-Formate)
- Ausführliches Tutorial unter `http://w3schools.com/xml/`

eXtensible Markup Language II

- XML-Dateien beginnen mit einem Header
- Daten sind zwischen XML-Tags eingeschlossen
- XML-Tags sind grundsätzlich nicht vordefiniert

Beispiel (`starwars.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
   <person>
      <name>Luke Skywalker</name>
      <homeworld>Tatooine</homeworld>
   </person>
   <person>
      <name>Han Solo</name>
      <homeworld>Corellia</homeworld>
   </person>
</people>
```

eXtensible Markup Language III

Beispiele (Attribute, Leere Elemente)

```
<person>
   <name>Han Solo</name>
   <homeworld>Corellia</homeworld>
   <height unit="meters">1.8</height>
</person>
```

```
<person>
   <name>Luke Skywalker</name>
   <jedi />
</person>
```

Zeichenvorrat I

An sich können sowohl in den Attributen als auch zwischen Tags beliebige Zeichen aus UTF-8 verwendet werden. Aber:

Entities

- & `&`
- < `<` (less than)
- > `>` (greater than)
- ' `&apos;` (apostrophe, single-quote)
- " `&quot;` (quotation mark, double-quote)

Beispiel

```
<text>7 &gt; 5</text>
```

Zeichenvorrat II

CDATA-Deklaration

Innerhalb von XML-Tags kann die CDATA-Deklaration verwendet werden, um anzuzeigen, dass Character-Daten verwendet werden. Dann müssen keine XML-Entities benutzt werden

Beispiel

```
<text><![CDATA[7 > 5]]></text>
```

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes: `http://www.unicode.org/charts/`

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes: `http://www.unicode.org/charts/`

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes: `http://www.unicode.org/charts/`

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes: `http://www.unicode.org/charts/`

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes:
  `http://www.unicode.org/charts/`

Leere Tags

Beispiele (Verschiedene Arten, leere Elemente zu notieren)

```
<jedi></jedi>
<jedi />
<jedi/>
```

Beispiel (Auch leere Tags können Attribute enthalten)

```
<jedi dark="no" />
```

Kommentare

Beispiel

```
<person>
   <name>Han Solo</name>
   <homeworld>Corellia</homeworld>
   <!-- dies ist ein kommentar -->
</person>
```

Markup/Datenformate

XML-Parser: DOM

- DOM steht für „Document Object Model”
- Liest das gesamte XML-Dokument in den Speicher und baut eine Baumstruktur im Speicher auf
- Objekte im DOM-Baum können über Methoden angesprochen werden (z.B. `childOf()`, `attributeOf()`)
- Vorteil: einfach und übersichtlich
- Nachteil: hoher Speicherverbrauch
- Weitere Informationen unter `http://www.w3.org/DOM/`

XML-Parser: SAX

- SAX steht für „Simple API for XML”
- Ereignis-basierte API: Man definiert aufzurufende Funktion, wenn bestimmte Elemente gefunden werden
- Nachteile: man muss die XML-Struktur gut kennen, keine automatischen Kontext-Informationen, Validierung sehr kompliziert
- Vorteile: sehr speichereffizient, sehr schnell
- Weitere Informationen unter `http://www.saxproject.org`

XML-Parsing in Python

Dokumentation

`https://docs.python.org/library/xml.html`

DOM in Python

- `xml.dom`
- `xml.etree.ElementTree`

XML-Parsing in Python

Dokumentation

`https://docs.python.org/library/xml.html`

DOM in Python

- `xml.dom`
- `xml.etree.ElementTree`

ElementTree Datenstrukturen

ElementTree überträgt das komplette XML-Dokument in eine Baumstruktur.

Element repräsentiert einen Knoten im XML-Baum mit folgenden Eigenschaften

- Tag als string
- Attribute als dictionary
- Text als string
- Alle Kindknoten als Sequenz-Objekt

XML-Parsing in Python

Beispiel (ElementTree)

```
>>> import pprint, xml.etree.ElementTree as et
>>> tree = et.parse('starwars.xml')
>>> r = tree.getroot()
>>> r.tag
'people'
>>> r.attrib
{}
>>> people = list(r) # Kindknoten von r
>>> person1 = people[0]
>>> pprint.pprint(list(person1)) # Kindknoten von person1
[<Element 'name' at 0x7fd824cc5150>,
 <Element 'homeworld' at 0x7fd824cc5210>,
 <Element 'jedi' at 0x7fd824cc52d0>]
 >>> people[0].find("name").text # find() Funktion
'Luke Skywalker'
```

XML-Parsing in Python

SAX

- Modul `xml.sax`
- in DOM wird ein Baumobjekt aus dem Dokument konstruiert.
- in SAX wird ein Parser-Objekt konstruiert, das das Dokument sequentiell nach bestimmten *Inhalten* durchsucht.
- Was mit den Inhalten gemacht werden soll, definiert ein `ContentHandler`, der dem Parser übergeben wird.
- SAX-Modul enthält einen Standard-ContentHandler, von dem ein eigener ContentHandler erbt.

XML-Parsing in Python

Beispiel (benutzerdefinierter ContentHandler)

```
import xml.sax

class JediSpotter(xml.sax.ContentHandler):
    def __init__(self):
        xml.sax.ContentHandler.__init__(self)

    def startElement(self,name,attrs):
        """überschreibt Methode von ContentHandler"""
        if name  == "jedi":
            print "I saw a jedi!"
```

XML-Parsing in Python

Beispiel (SAX Parsing)

```
>>> handler = JediSpotter()
>>> xml.sax.parse(open('starwars.xml'),handler)
I saw a Jedi!
```

XML und HTML

Historisch gesehen kommt XML nach HTML, ist aber allgemeiner und hat eine andere Zielsetzung:

- HTML beschreibt, wie Daten *aussehen* sollen
- XML beschreibt, welche Informationen Daten beinhalten und welche Eigenschaften sie haben

HTML - Beispiel

Beispiel

```
<b>
   Fetter Text.
</b>
Das ist ein
<a href="http://www.google.de">
   Link
</a>
```

Ausführliche Informationen zu HTML: `http://de.selfhtml.org`

Markup/Datenformate

XPath I

- XML-Strukturen sind baumartig
- XPath beschreibt Pfade durch diesen XML-Baum
- Die XPath-Notation ist Pfaden im Unix-Dateisystem sehr ähnlich
- Für XPath ist jede Entität in XML ein Knoten:
  - Elemente
  - Attribute
  - Text
  - Kommentare
  - ...
- Weitere Informationen unter `http://www.w3schools.com/xpath/`

XPath II

Beispiel

```
<people>
   <person>
      <name>Luke Skywalker</name>
   </person>
   <person>
      <name>Han Solo</name>
   </person>
</people>
```

`/people`	people
`/people/person`	person (Wählt mehr als ein Element aus!)
`/people/person[1]`	person (wählt das erste person unter people)

XPath III

Syntax

knotenname	Wählt alle (direkten) Kindknoten mit Tag *knotenname*
/	Pfad beginnt am Root-Knoten
//	Wählt Knoten im Dokument unabhängig davon, wie tief sie eingebettet sind.
.	Wählt den aktuellen Knoten
..	Wählt den Eltern-Knoten des aktuellen Knotens
@	Wählt Attribute

XPath IV

Beispiele

`people`	Wählt alle direkten Kindknoten mit Tag `people`
`/people`	Wählt das root-Element people
`//person`	Wählt alle person-Elemente, egal wo sie im Dokument stehen
`//people/person`	Wählt alle person-Elemente, die Kind von einem people-Element sind
`//@unit`	Wählt alle Attribute, die unit heißen
`/people/person[last()]`	Wählt das letzte person-Element unter people
`//height[@unit='meters']`	Wählt alle height-Elemente, die ein Attribut unit haben, das auf „meters" gesetzt ist

Beispiele Python

- `ElementTree`: teilweise Unterstützung von XPath Syntax
- Suchpfad muss immer relativ zum aktuellen Knoten angegeben werden (siehe Beispiel)!

```
>>> import xml.etree.ElementTree as et
>>> tree = et.parse('starwars.xml')
>>> r = tree.getroot()
>>> # alle name-Elemente
>>> r.findall(".//name")
```

https://docs.python.org/2/library/xml.etree.elementtree.html#xpath-support

Markup/Datenformate

JSON

- JSON = `JavaScript Object Notation`
- Textbasiertes Format zum Datenaustausch.
- Entwickelt in den frühen 2000ern für Kommunikation zwischen Server und Webanwendung.
- Verwendet JavaScript(-ähnliche) Syntax.
- In vielen Programmiersprachen sind Bibliotheken für Dekodierung und Generierung von JSON vorhanden

`http://www.w3schools.com/json/`

Vergleich mit XML

Gemeinsamkeiten

- Textbasiert, für Menschen (und Maschinen) lesbar
- Hierarchisch
- Sprachunabhängig

Unterschiede

- XML ist eine Auszeichnungssprache
- JSON ist ein Datenformat
- JSON kann Arrays verwenden

Beispiel

Beispiel (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
   <person>
      <name>Luke Skywalker</name>
      <homeworld>Tatooine</homeworld>
   </person>
   <person>
      <name>Han Solo</name>
      <homeworld>Corellia</homeworld>
   </person>
</people>
```

Syntax I

Schlüssel-Wert-Paare

- Daten werden als Schlüssel-Wert-Paare dargestellt (`"name":"Han Solo"`)
- Schlüssel ist immer eine Zeichenkette
- Mögliche Werte:
  - null
  - Zahl (`int`,`float`)
  - Zeichenkette
  - Wahrheitswert
  - Array (in `[]`)
  - Objekt

Syntax II

JSON-Objekte

- Stehen in `{ }`
- Enthalten durch Kommata getrennte Schlüssel-Wert-Paare
- In einem Objekt darf jeder Schlüssel nur einmal verwendet werden!

Beispiel (JSON)

```
{"people":[
    {"name":"Luke Skywalker", "homeworld":"Tatooine"},
    {"name":"Han Solo", "homeworld":"Corellia"}
]}
```

Warum JSON?

- Kürzer als XML (keine End-Tags), daher weniger Speicherverbrauch
- Oft weniger Overhead beim Einlesen von JSON-Daten
- Häufig verwendet: z.B. Twitter API, Serialisierung von Datenstrukturen
- Nützlich für weniger komplexe Daten (z.B. Metadaten)

JSON verwenden

Beispiel (JSON in Python)

```
>>> import json, pprint
>>> obj = json.load(open('starwars.json'))
>>> pprint.pprint(obj)
{u'people': [{u'homeworld': u'Tatooine',
              u'name': u'Luke Skywalker'},
             {u'homeworld': u'Corellia',
              u'name': u'Han Solo'}]}
>>> json.dump(obj,open('starwars_dump.json','w'),indent=3)
```

https://docs.python.org/2/library/json.html

Übung 5

Mittagspause

Dienstag: Versionskontrolle und Makefiles

5 Versionskontrolle und Makefiles
- Versionskontrolle
- Makefiles

Versionskontrolle und Makefiles

Projekte

- Dauern lang
- Involvieren mehrere Personen und
- Verzweigen manchmal

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Versionsverwaltung

- Protokollieren alle Änderungen
- Verhindern das gegenseitige Überschreiben
- Können automatisch Änderungen von verschiedenen Leuten an der gleichen Datei zusammenführen (manchmal)
- Stellen Unterschiede zwischen Versionen dar

Systeme

Es gibt eine Vielzahl von Versionsverwaltungssystemen:

- CVS (Concurrent Versioning System)
- BitKeeper (proprietär, wurde vom Linux-Kernel verwendet)
- Bazaar (Wird von der Ubuntu-Distribution verwendet)
- SVN (Subversion, als Ersatz für CVS entwickelt)
- git (frei, wird heute für den Linux-Kernel verwendet)

Systeme

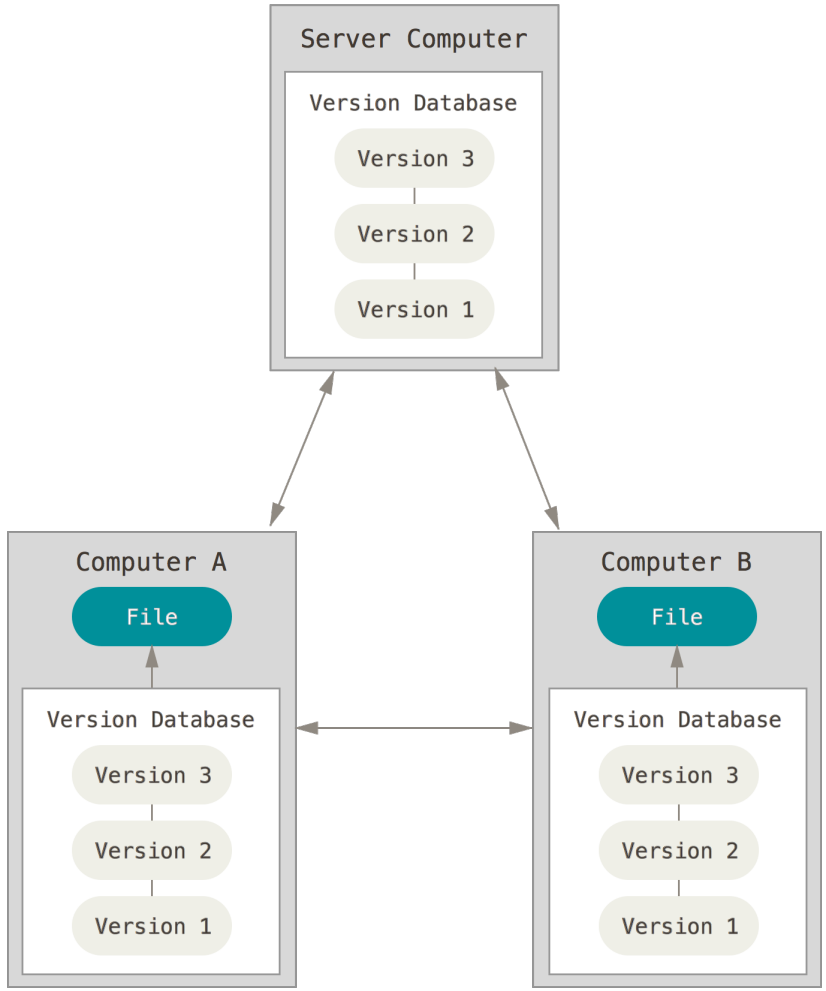
Es gibt eine Vielzahl von Versionsverwaltungssystemen:

- CVS (Concurrent Versioning System)
- BitKeeper (proprietär, wurde vom Linux-Kernel verwendet)
- Bazaar (Wird von der Ubuntu-Distribution verwendet)
- SVN (Subversion, als Ersatz für CVS entwickelt)
- git (frei, wird heute für den Linux-Kernel verwendet)

git

- *verteiltes* System: jeder Benutzer besitzt eine Kopie des gesamten Repositorys
  - stets Zugriff auf komplette Versionsgeschichte
  - ermöglicht *Offline*-Entwicklung
- Dokumentation: `https://git-scm.com/doc`
- man arbeitet stets in seiner Kopie des Repositorys

Übersicht



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Repository

- Man kann entweder selber ein Repository erstellen:

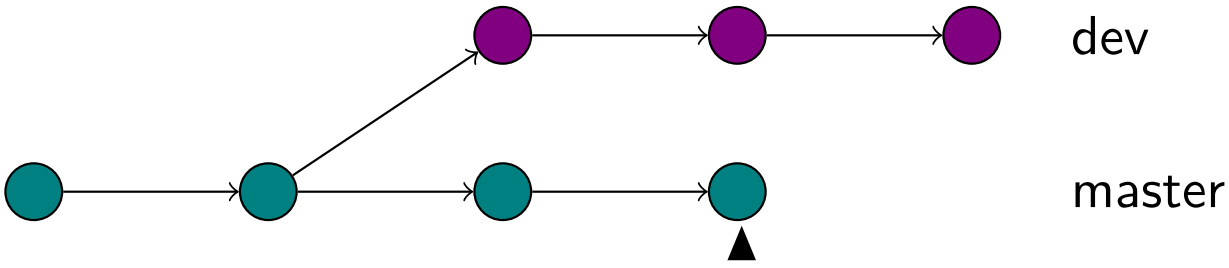
```
:~$ git init <folder>
```

- Oder man “klont” ein vorhandenes Repository:

```
:~$ git clone <url>
```

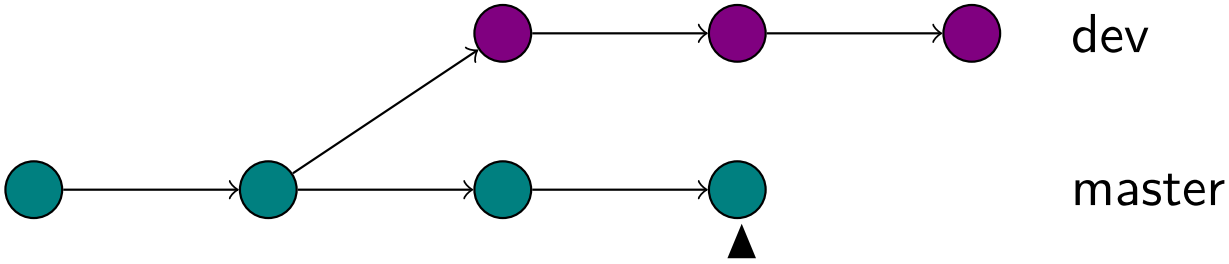
Visualisierung

Remote auf dem Server (**`origin`**):

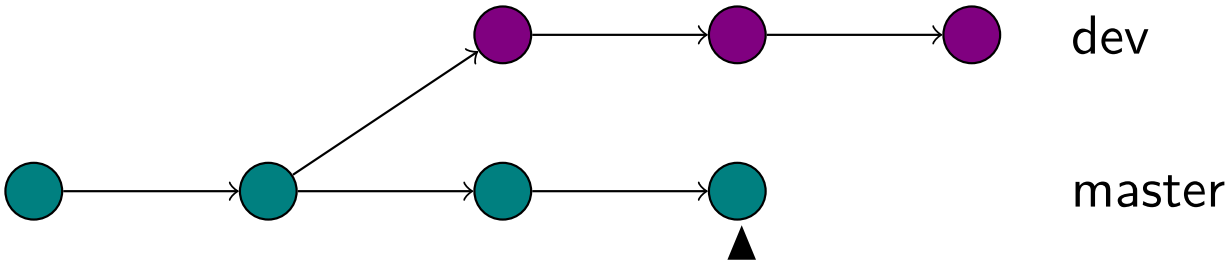


Visualisierung

Remote auf dem Server (**`origin`**):



Lokal nach **`git clone`**:



Paradigma

- Man arbeit in der lokalen Kopie des Repositorys
- Dateien, die man geändert oder hinzugefügt hat, markiert man mit `:~$ git add <filename>`
- Um die so markierten Änderungen zu speichern, benutzt man dann `:~$ git commit`
  - eine Protokollnachricht **muss** angegeben werden! (Mit `-m MESSAGE` oder in einem Editor)
  - `:~$ -a` fügte bekannte Dateien automatisch zum Commit hinzu

Protokollnachrichten

- Zu jeder Revision eine Protokollnachricht (*log message*)
- Es erhöht die Übersichtlichkeit beträchtlich, etwas sinnvolles anzugeben
- gut: “Methode Klasse123.getABC() hinzugefügt”, “Fehler 123 gefixt”
- schlecht: “Änderungen gemacht”, “Klasse123 repariert”

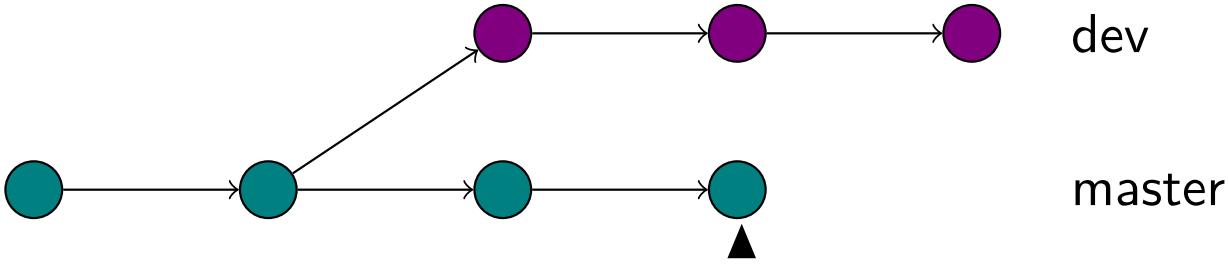
remove

```
:~$ git rm <filename>
```

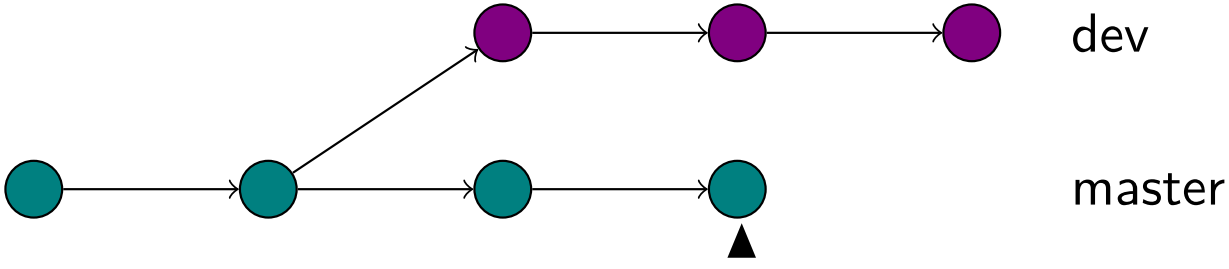
- Dateien unter Versionskontrolle dürfen **nicht** einfach gelöscht werden!
- Dateien müssen per git gelöscht werden

Visualisierung: Comitten

Remote auf dem Server (**`origin`**):

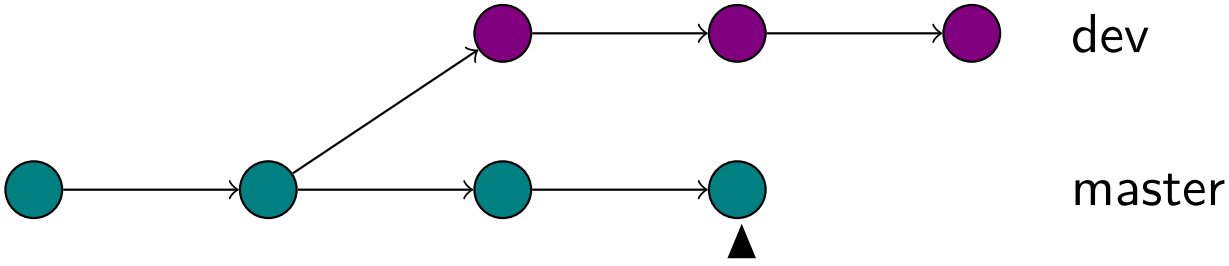


Lokal vor **`git commit`**:

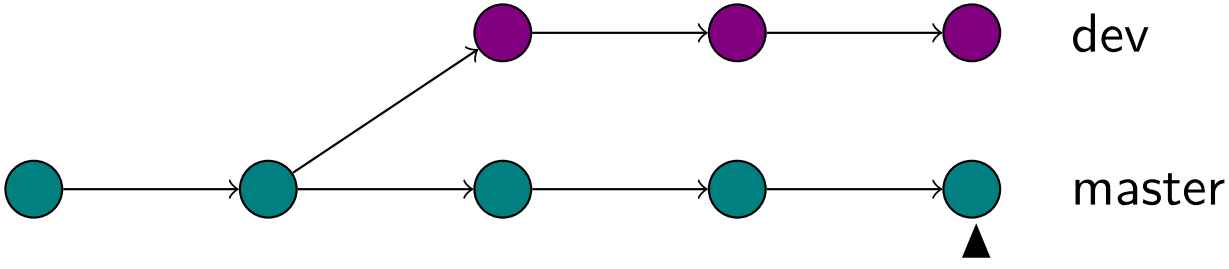


Visualisierung: Comitten

Remote auf dem Server (**`origin`**):



Lokal nach **`git commit`**:



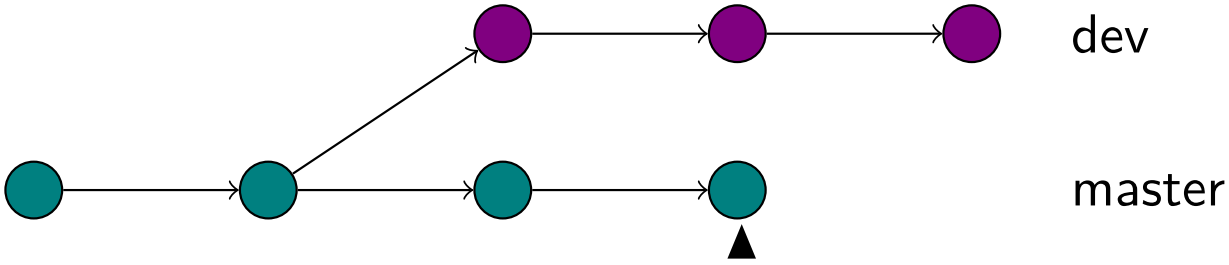
push

```
:~$ git push <remote>
```

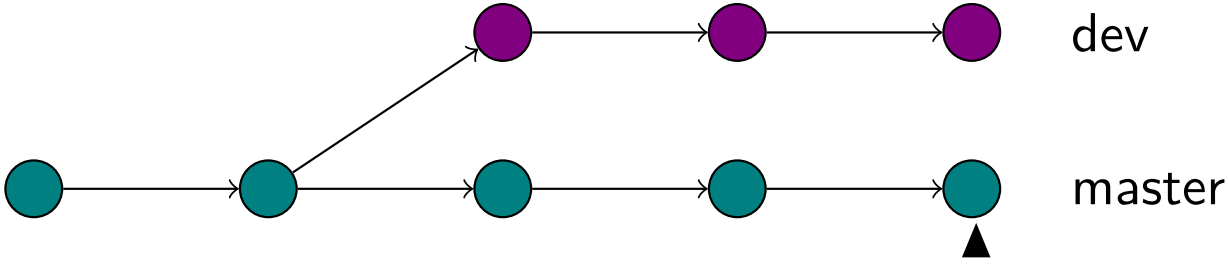
- aktualisiert das Repository names **<remote>** mit dem Status der lokalen Version des Repositorys
- default: **origin**
- Dateikonflikte sind möglich, falls jemand anderes schon etwas aktualisiert hat – dazu kommen wir später

Visualisierung: push

`origin` vor `git push origin`:

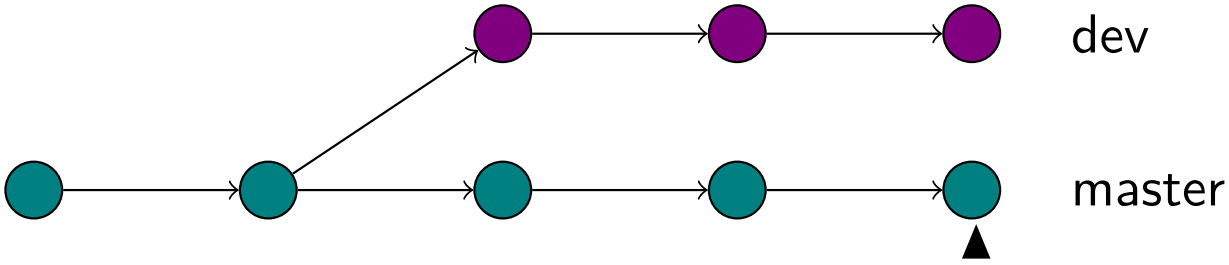


Lokal:

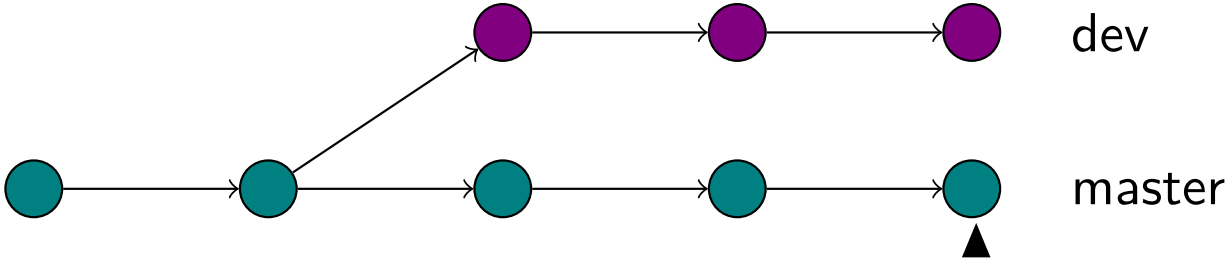


Visualisierung: push

`origin` nach `git push origin`:



Lokal:



fetch

```
:~$ git fetch <remote>
```

- aktualisiert lokale Information über **<remote>**
- Dateien werden noch nicht geändert!

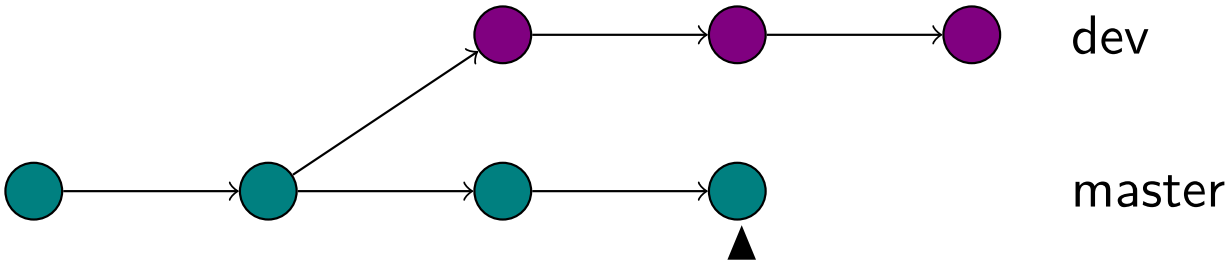
merge

```
:~$ git merge <remote>/<branch>
```

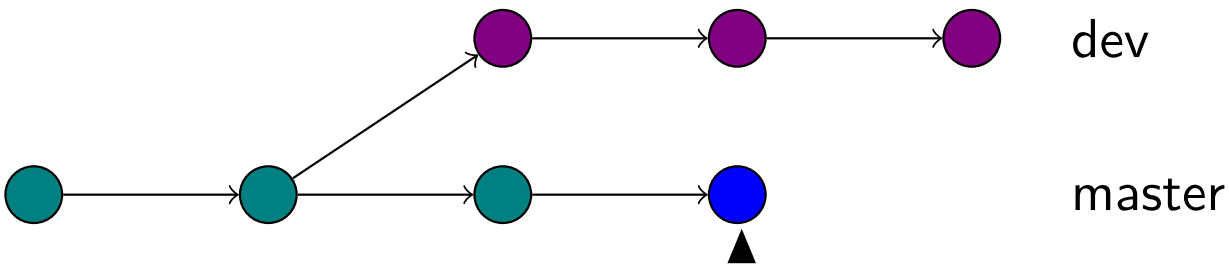
- fügt angegebenen Branch und aktuellen zusammen
- eventuell müssen Dateikonflikte aufgelöst werden
- falls keine Konflikte: es wird automatisch comittet!

Visualisierung: fetch und merge

`origin`:

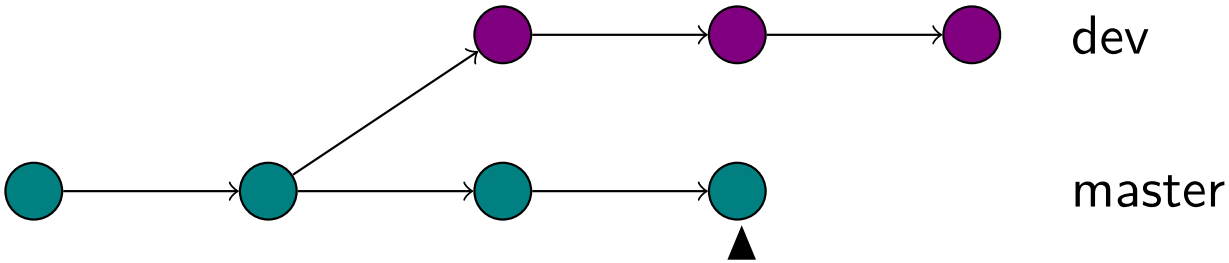


Lokal vor `git fetch origin`:

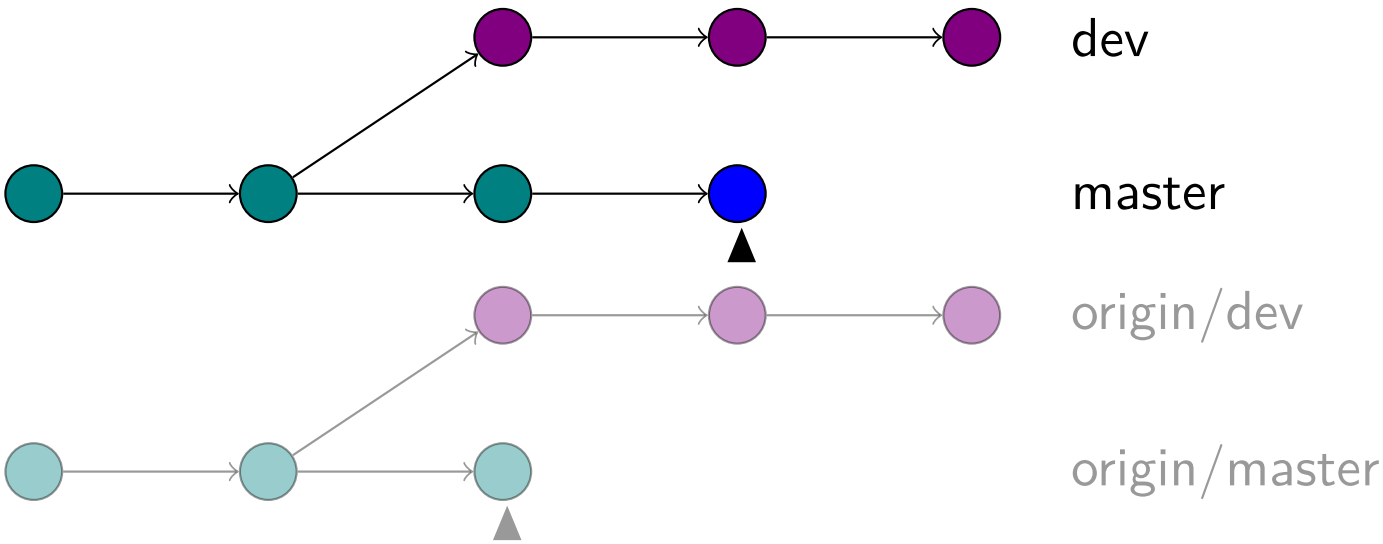


Visualisierung: fetch und merge

`origin`:

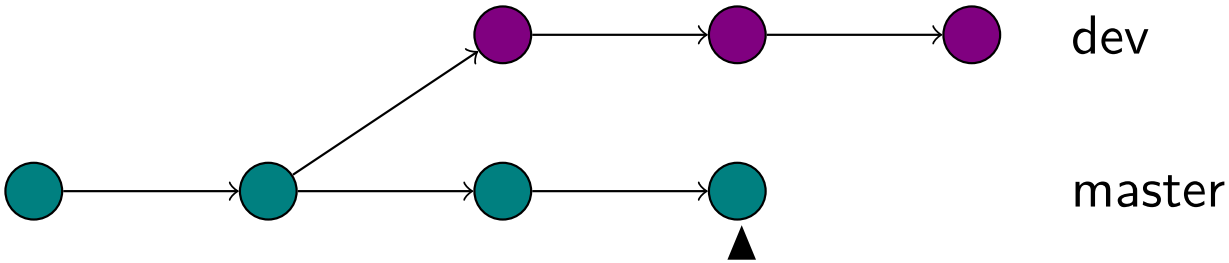


Lokal vor `git fetch origin`:

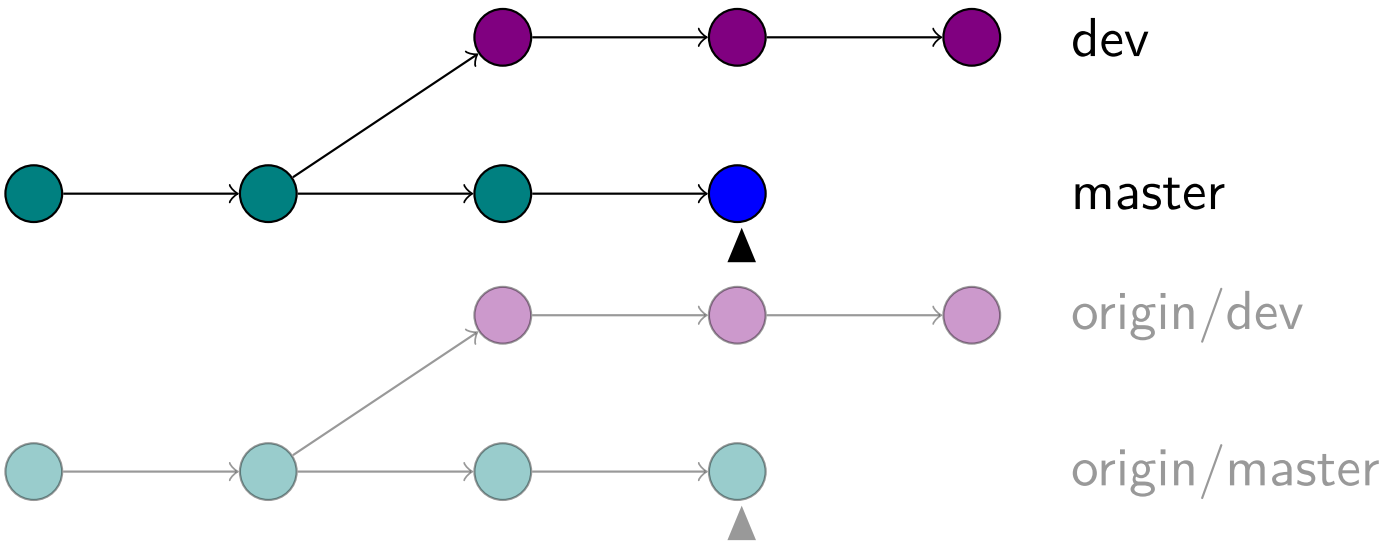


Visualisierung: fetch und merge

`origin`:

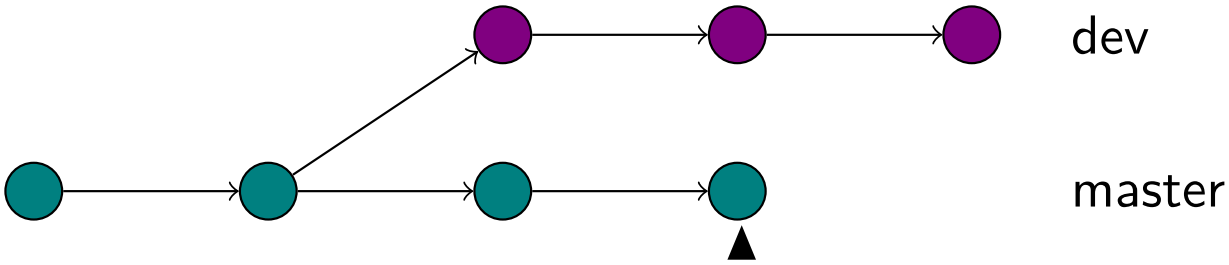


Lokal nach `git fetch origin`:

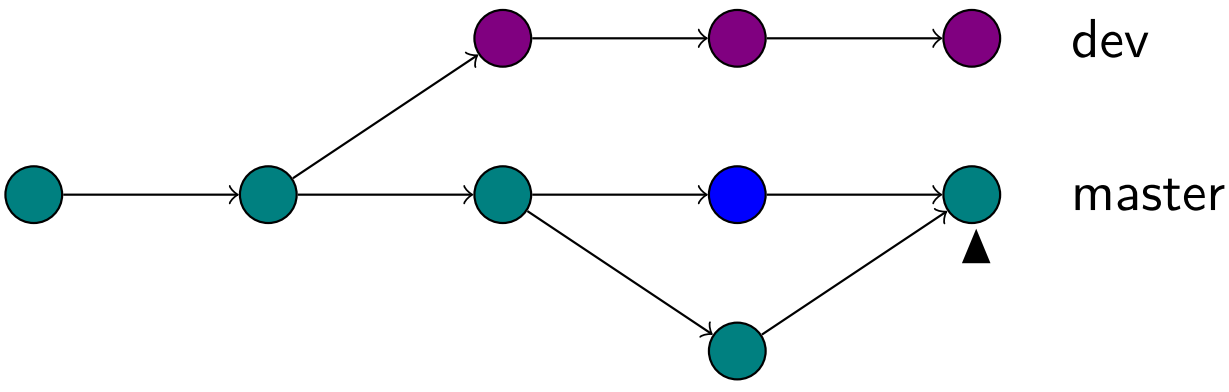


Visualisierung: fetch und merge

`origin`:



Lokal nach `git merge origin/master`:



pull

```
:~$ git pull <remote>/<branch>
```

- ruft zunächst `:~$ git fetch` auf, danach `:~$ git merge`

Branchmanagement

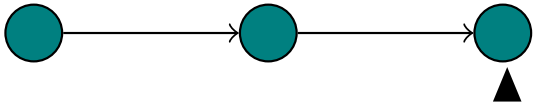
- Manchmal möchte man komplexe Dinge ausprobieren, die nicht mit einer Änderung getan sind
- Dafür kann man einen Branch anlegen
- `:~$ git branch <branch>` legt Branch namens **<branch>** an
- `:~$ git checkout <branch>` wechselt in Branch **<branch>**
- `:~$ git branch -d <branch>` löscht Branch **<branch>**
- der "Standardbranch" ist **master**

Merging von Branches

- Workflow:
  1. `:~$ git branch <branch>`
  2. `:~$ git checkout <branch>`
  3. Arbeiten durchführen und comitten
  4. `:~$ git checkout master`
  5. `:~$ git merge <branch>`
  6. `:~$ git branch -d <branch>`
- Dateikonflikte können auftreten, falls an dem Branch, von dem abgezweigt wurde, weitergearbeitet wurde

Visualisierung: Branches

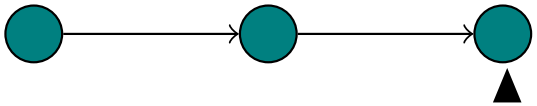
Lokal ursprünglich:



master

Visualisierung: Branches

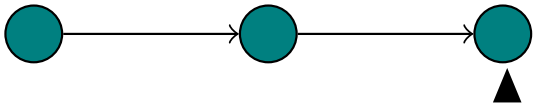
Lokal nach `git branch dev`:



master, dev

Visualisierung: Branches

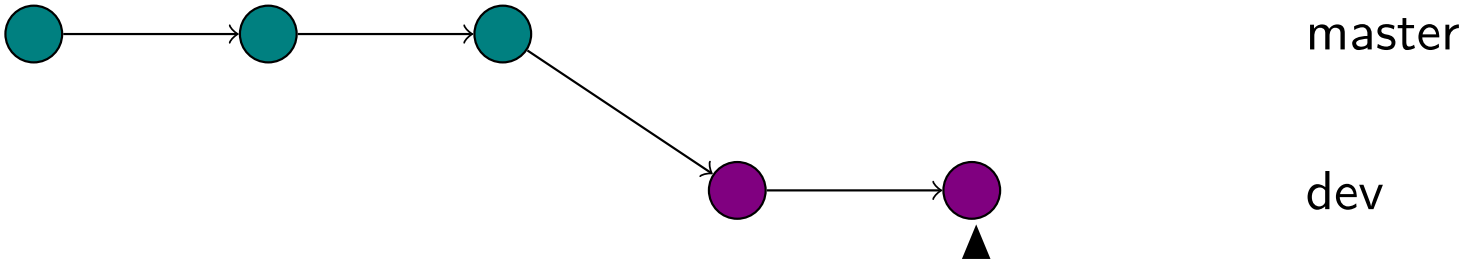
Lokal nach `git checkout dev`:



master, dev

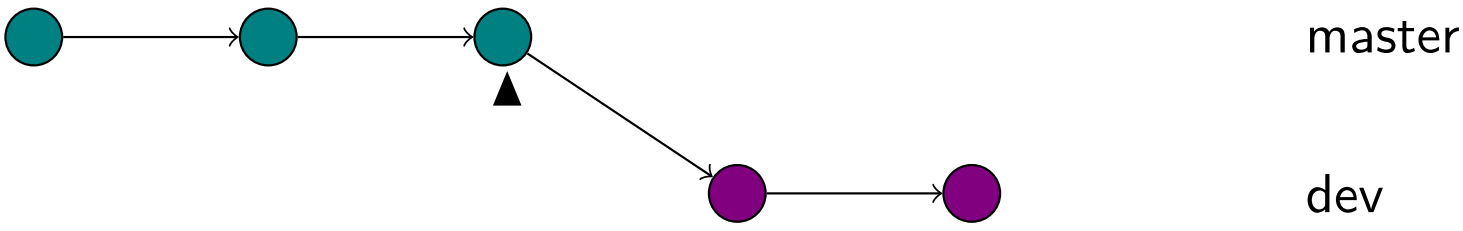
Visualisierung: Branches

Lokal nach Änderungen und mehreren `git commit`:



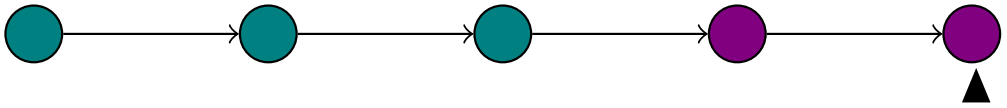
Visualisierung: Branches

Lokal nach `git checkout master`:



Visualisierung: Branches

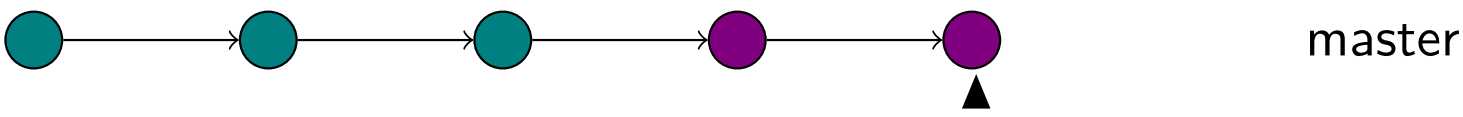
Lokal nach `git merge dev`:



master, dev

Visualisierung: Branches

Lokal nach `git branch -d dev`:



Dateikonflikte auflösen I

- Beim Arbeiten mit verschiedenen Branches und bei der Interaktion mit anderen Versionen des Repositorys können Dateikonflikte auftreten
- `git status` – Konflikte werden angezeigt
- Entweder:
  - in die Datei gehen, Konflikte manuell auflösen oder
  - `git mergetool` benutzen

Dateikonflikte auflösen II

- `:~$ git add <filename>` ausführen, um anzuzeigen, dass die Konflikte aufgelöst wurden
- Zuletzt committen:

```
:~$ git commit -m 'Resolved conflict between ...'
```

Hilfe und Status

```
:~$ git help
```

- Zeigt eine Übersicht der verfügbaren Befehle an

```
:~$ git help <command>
```

- Zeigt Hilfe zu bestimmten Kommandos an

```
:~$ git status
```

- Zeigt den aktuellen Status an

Log und Blame

```
:~$ git log
```

- Gibt commit-Historie aus
- Listet Author, Datum und Commit-Message
- Commit wird über Hash indentifiert
  - Wird aus den Inhalten des Commits berechnet
  - Normalerweise reichen die ersten sieben Stellen, um einen commit commit zu identifizieren

```
:~$ git blame <file>
```

- Listet für jede Zeile in `<file>` auf, wer sie wann zuletzt geändert hat
- `git blame -L<start>,<stop> <file>` zeigt nur Zeilen zwischen `start` und `stop`

Alte Versionen betrachten

```
:~$ git diff
```

- Zeigt Änderungen, die noch nicht im Index sind (nicht geadded)

```
:~$ git diff <commit_hash> <filename>
```

- Zeigt Unterschied zwischen momentanem Zustand und früherem Commit

```
:~$ git show <commit_hash>:<filename>
```

- Zeigt alte Version einer Datei an

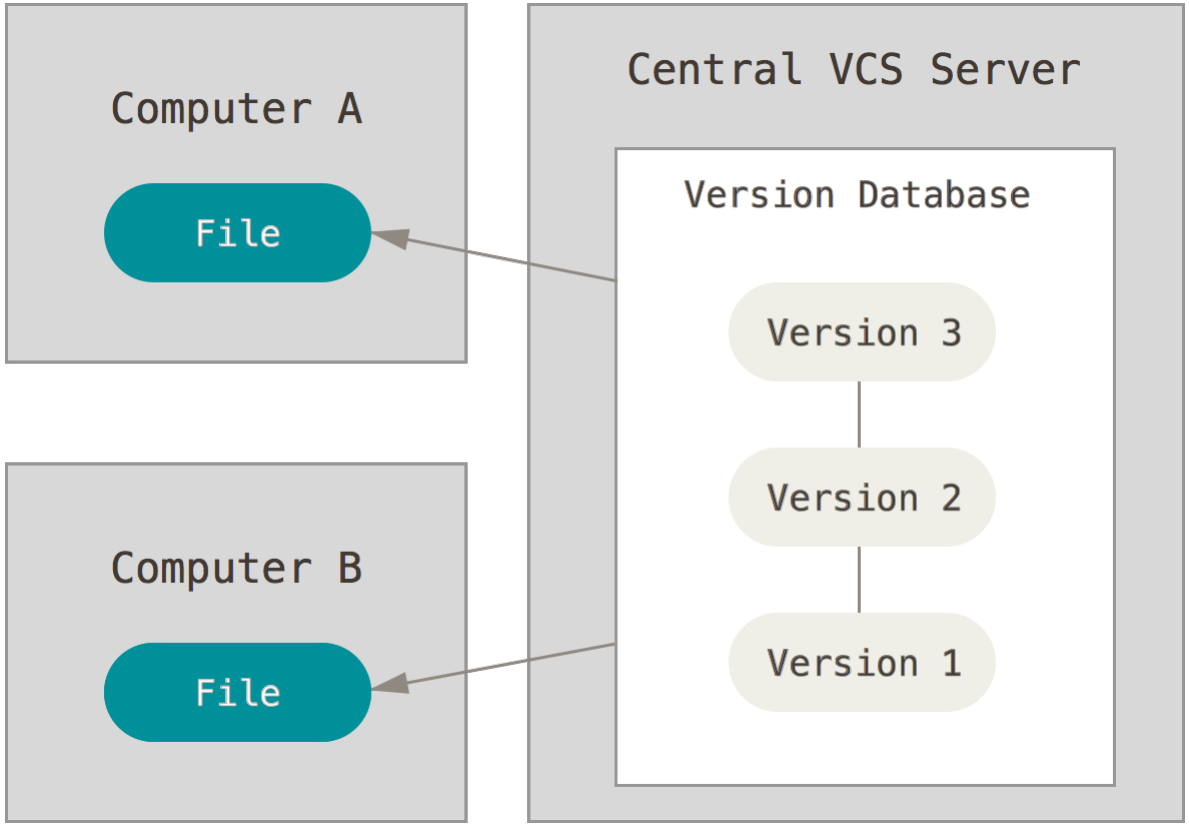
Unerwünschte Dateien

- Manche Dateien gehören nicht in ein Repository
  - Kompilierte Dateien (`.pyc`)
  - Potenziell vertraunliche Dateien (Konfigurationen mit Passwörtern)
- `.gitignore`-Datei kann diese Dateien automatisch ausschließen
- Datei liegt im root-Verzeichnis des Repository
- Enthält Patterns von unerwünschten Dateien (`*.pyc`)
- Nützliche Ressource für vorgefertigte Dateien: `https://github.com/github/gitignore`

Subversion

- Zentralisiertes System
- Eine Versionsnummer für das ganze Repository
- Dokumentation: `http://svnbook.org`
- Man arbeitet NIE im Repository

Übersicht



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Subversion: Befehle

- Größter Unterschied: git ist verteilt, svn ist zentralisiert
- Viele Ähnliche Befehle wie git, aber mit anderer Bedeutung
  - `svn checkout` ist ungefähr `git clone`
  - `svn commit` kombiniert `git add`, `git commit` und `git push`

Übung 6

Versionskontrolle und Makefiles

Building

- Beim *Build-Prozess* (Erstellungsprozess) wird Text in Binärdateien konvertiert.
- Meistens: Quellcode in selbständiges Programm kompilieren.
- Aber auch: PDF aus LaTeX-Dokument erstellen.
- Und: Archiv aus einem Projekt erstellen
- Im Allgemeinen: Eine Menge von Anweisungen, die oft in einer bestimmten Reihenfolge ausgeführt werden (z.B. `pdflatex, bibtex, pdflatex, pdflatex`)

Abhängigkeiten

- Erstellungsschritte müssen in einer bestimmten Reihenfolge ausgeführt werden.
- Haben sich nur Teile des Projekts geändert, müssen nur diese neu erstellt werden.
  1. file1.c → file1.o
  2. file2.c → file2.o
  3. file1.o, file2.o → program

Abhängigkeiten

- Erstellungsschritte müssen in einer bestimmten Reihenfolge ausgeführt werden.
- Haben sich nur Teile des Projekts geändert, müssen nur diese neu erstellt werden.
  1. file1.c $\rightarrow$ file1.o
  2. file2.c $\rightarrow$ file2.o
  3. file1.o, file2.o $\rightarrow$ program

Abhängigkeiten

- Erstellungsschritte müssen in einer bestimmten Reihenfolge ausgeführt werden.
- Haben sich nur Teile des Projekts geändert, müssen nur diese neu erstellt werden.
  1. file1.c $\rightarrow$ file1.o
  2. file2.c $\rightarrow$ file2.o
  3. file1.o, file2.o $\rightarrow$ program

Abhängigkeiten

- Erstellungsschritte müssen in einer bestimmten Reihenfolge ausgeführt werden.
- Haben sich nur Teile des Projekts geändert, müssen nur diese neu erstellt werden.
  1. file1.c $\rightarrow$ file1.o
  2. file2.c $\rightarrow$ file2.o
  3. file1.o, file2.o $\rightarrow$ program

Abhängigkeiten

- Erstellungsschritte müssen in einer bestimmten Reihenfolge ausgeführt werden.
- Haben sich nur Teile des Projekts geändert, müssen nur diese neu erstellt werden.
  1. file1.c $\rightarrow$ file1.o
  2. file2.c $\rightarrow$ file2.o
  3. file1.o, file2.o $\rightarrow$ program

Versionskontrolle und Makefiles

Make

- Build-Management Tool
- Ermöglicht Definition von Abhängigkeiten
- Abhängigkeiten und Anweisungen werden in `Makefiles` geschrieben.
- Dokumentation:
  `http://www.gnu.org/software/make/manual/`

```
:~$ make <target>
```

Makefile

Struktur

```
target: prerequisities
	command
	command
```

Jede `command`-Zeile muss mit einem *tab* eingerückt sein!

Bestandteile

target (Ziel) oft Name der erzeugten Datei (z.B. `slides.pdf`)

prerequisites (Bedingungen) Dateien (oder andere Ziele), die im Voraus benötigt werden (z.B. `slides.tex`)

command (Anweisung) Wie ausgeführt, um Ziel zu erstellen (z.B. `pdflatex`)

Makefile-Beispiel

```
all: Token.class Tagger.class

# This is a comment
Token.class: Token.java
        javac Token.java

Tagger.class: Tagger.java
        javac Tagger.java

javadoc: Tagger.java Token.java
        javadoc *.java
```

Variablen

- Bedingungen können Variablen enthalten
- Diese werden am Anfang deklariert:

```
JAVAC=javac
```

- Und wie folgt verwendet:

```
Token.class: Token.java
        ${JAVAC} Token.java
```

Pattern-Regeln

- Bei größeren Projekten enthalten Makefiles oft viele Redundanzen.
- Pattern-Regeln definieren eine Regel für alle Dateien vom selben Typ.
- Pattern-Regel für Kompilierung von java Dateien:

```
%.class : %.java
	javac $<
```

(Einige) automatische Variablen

`$@`	Ziel der Regel
`$<`	Die erste Bedingung
`$?`	Alle Bedingungen, die neuer als das Ziel sind
`$^`	Alle Bedingungen

Mehr: `http://www.gnu.org/software/make/manual/html_node/Automatic-Variables.html`

Versionskontrolle und Makefiles

Apache Ant

- Neueres *Build*-Werkzeug
- Konzentriert auf Java
- **`Makefile`** $\rightarrow$ **`build.xml`**
- XML-basiert
- Dokumentation: `http://ant.apache.org/manual/`

ant ausführen

- Kommandozeile: `:~$ ant [-buildfile file.xml]`
- Wenn `-buildfile` nicht angegeben ist, sucht `ant` im momentanen Verzeichnis nach einer Datei `build.xml`
- Option `-find [file]`: ant durchsucht alle übergeordneten Verzeichnisse nach `build.xml`
- Target angeben: `:~$ ant -buildfile file.xml target`
- Weitere Optionen `:~$ ant -help`

build.xml - Struktur

```
<project name="MyProj" default="all" basedir=".">
  <!--- optional: project description -->
    <description>
        simple example build file
    </description>

    <!-- set global properties for this build -->
    <property name="src" location="src"/>
    <property name="build" location="build"/>
    <property name="dist"  location="dist"/>

    <!-- targets -->

</project>
```

build.xml - Targets

```
<target name="init">
   <!-- Create the build directory structure used by compile -->
   <mkdir dir="${build}"/>
   <mkdir dir="${dist}" />
</target>

<target name="compile" depends="init"
        description="compile the source">
    <!-- Compile the java code from ${src}
         into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
</target>
```

Classpath

- Classpath kann im `javac`-Element spezifiziert werden.

```
<classpath>
    <pathelement location="lib/tagger.jar" />
    <pathelement location="lib/helper.jar" />
</classpath>
```

Wichtigste Merkmale

- Jede `build.xml` enthält genau ein `project`
- Jedes Projekt definiert mindestens ein `target` (eine Menge von *Tasks*)
- Task sind ausführbare Statements, z.B. `mkdir`, `javac`. Liste der Builtin-Tasks:
  `http://ant.apache.org/manual/tasklist.html`
- *Properties* haben eine ähnliche Rolle wie Variablen bei `make`.

Vorteile von `ant`

- Plattformunabhängig
- Keine Tabs vor Befehlen $\rightarrow$ Weniger fehleranfällig.

Übung 7