UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

INSTITUT FÜR COMPUTER-LINGUISTIK

Einführung in die Nutzung computerlinguistischer Ressourcen

Juri Opitz

15.10.2019

Material in Teilen von Laura Jehl, Nils Reiter, Shigehiko Schamoni,Sebastian Martschat und Julius Steen

Organisatorisches

- Ziel: Vertrautheit mit Daten und Tools zur Bearbeitung von Softwareprojekten erlangen
- Integrierte praktische Übungen
- Zeiten: Montag bis Freitag, 10:00 Uhr bis 12:00 Uhr und 13:00 Uhr bis 18:00 Uhr
- Slides und Übungen verfügbar unter `http://cl.uni-heidelberg.de/courses/ws19/ressourcen/`
- falls Sitzung verpasst: Übungen bitte nachträglich bearbeiten und am `opitz@cl.uni-heidelberg.de` senden

Der Plan

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
Vormittag	Linux I	Korpora, Formate	Parser, Tagger	Python für ML	Arbeiten mit dem Cluster
Nachmittag	Linux II	VCS, Makefiles	Python für NLP	Weka	

Montag: Linux I

1. Linux I
   - Grundlagen
   - Prozesse, Streams und Pipes
   - Reguläre Ausdrücke
   - Batch-Verarbeitung

Linux I

Motivation

- viele Tools der Computerlinguistik sind für Linux entwickelt
- Linux stellt mächtige Kommandozeilentools zur Verfügung
- nützlich für automatisierte
  - Datenorganisation
  - Datenaufbereitung
  - Extraktion von Informationen

Terminal

Sinn und Zweck

- Benutzerschnittstelle zu Computern mit Unix-Systemen (z.B. Linux, Mac OS X)
- Befehle werden textuell eingegeben und *nacheinander* verarbeitet

Terminal vs. Shell

Terminal Ein *Gerät*, das Interaktion mit einem Computer erlaubt

Shell Ein *Programm*, das auf einem Computer läuft, Befehle entegegennimmt und interpretiert

Terminals I


ASR-33 Teletype
• Tymeshare/Tymnet era
• Stand
• Cover
• Original paper roll
• Complete 3-binder manual set
• Powers up, but print head doesn't advance and ribbon needs replacing or re-inking
$600.00
A1
TELETYPE
COMP LCL OFF
TYMSHARE
TIME SHARING TERMINAL
TELETYPE CORPORATION

Terminals II

Terminalemulator

- Terminalemulatoren “emulieren” (täuschen vor) ein Terminal. Sie laufen in graphischen Umgebungen in eigenen Fenstern.
- Innerhalb des Fensters laufen die gleichen Programme, die auch in echten Terminals laufen (eine Shell).
- Heute lässt man “-emulatoren” oft weg und spricht einfach von Terminal.

Shell

- Die Shell läuft innerhalb des Terminals
- Es gibt viele unterschiedliche Shells.
- Wir benutzen ausschließlich die Bourne-Again shell: bash.

Befehle

- Befehle werden in Textform eingegeben und der Reihe nach ausgeführt.
- Befehle haben
  - einen Namen
  - Optionen (*wie der Befehl ausgeführt wird*)
  - Argumente (*worauf der Befehl angewendet wird*)
  - Optionen und Argumente sind Parameter eines Befehls
- Befehle sind eigene Programme (z.B. `less`, `wc`) oder Builtins, also Teile der Shell (z.B. `cd`, `echo`).

Hilfe zur Selbsthilfe

- Man weiß nie alle Optionen auswendig
- Wichtig: Man kann Optionen nachschauen
- Viele Programme zeigen selbständig Hilfe an: **`-h`**, **`-help`** oder **`--help`**
- Viele Programme bieten eine Man-page an:

  ```
  :~$ man <Programmname>
  ```

- Hilfe zur Kommandozeile insgesamt:
  `http://tldp.org/LDP/abs/html/` oder

  ```
  :~$ man bash
  ```

Notation

Einzelne Befehle

- sind in **`Maschinenschrift`** gesetzt

Vollständige Eingaben

- haben einen Prompt, sind in `Maschinenschrift` und grau hinterlegt

```
:~$ Befehl
```

Grundlegende Befehle

Navigieren im Verzeichnisbaum

`ls`	Zeigt Verzeichnisinhalt an
`cd`	Wechselt das aktuelle Verzeichnis
`pwd`	Gibt das aktuelle Verzeichnis aus
`mkdir`	Legt ein neues Verzeichnis an
`rmdir`	Löscht ein leeres Verzeichnis

Hilfe zur Selbsthilfe

`man`	Zeigt manual an
`info`	Zeigt info-Seiten an

Grundlegende Befehle

Navigieren im Verzeichnisbaum

`ls`	Zeigt Verzeichnisinhalt an
`cd`	Wechselt das aktuelle Verzeichnis
`pwd`	Gibt das aktuelle Verzeichnis aus
`mkdir`	Legt ein neues Verzeichnis an
`rmdir`	Löscht ein leeres Verzeichnis

Hilfe zur Selbsthilfe

`man`	Zeigt manual an
`info`	Zeigt info-Seiten an

Lange und kurze Optionen

Die meisten Programme folgen gewissen Konventionen in der Interpretation ihrer Eingaben.

Kurze Optionen

- Beginnen mit einem einfachen Strich
- Sind nur ein Zeichen lang: `-l`
- Mehrere können zusammengefasst werden: `-la` statt `-l -a`

Lange Optionen

- Beginnen mit zwei Strichen
- Sind länger als ein Zeichen: `--help`

- Viele Optionen gibt es sowohl in kurz als auch in lang (man page)

Lange und kurze Optionen

Die meisten Programme folgen gewissen Konventionen in der Interpretation ihrer Eingaben.

Kurze Optionen

- Beginnen mit einem einfachen Strich
- Sind nur ein Zeichen lang: `-l`
- Mehrere können zusammengefasst werden: `-la` statt `-l -a`

Lange Optionen

- Beginnen mit zwei Strichen
- Sind länger als ein Zeichen: `--help`

- Viele Optionen gibt es sowohl in kurz als auch in lang (man page)

Lange und kurze Optionen

Die meisten Programme folgen gewissen Konventionen in der Interpretation ihrer Eingaben.

Kurze Optionen

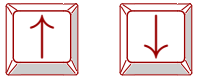
- Beginnen mit einem einfachen Strich
- Sind nur ein Zeichen lang: `-l`
- Mehrere können zusammengefasst werden: `-la` statt `-l -a`

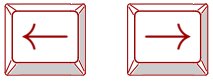
Lange Optionen

- Beginnen mit zwei Strichen
- Sind länger als ein Zeichen: `--help`

- Viele Optionen gibt es sowohl in kurz als auch in lang (man page)

Tastenkürzel


Blättern in der Befehlshistorie

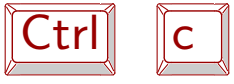

Editieren des aktuellen Befehls


Automatisches Ergänzen von Datei- und Verzeichnisnamen

Springe zum Anfang der Zeile

Springe zum Ende der Zeile

Suche in der Befehlshistorie



Bricht den gerade laufenden Prozess ab

Mehr Befehle

`less`	Zeigt den Inhalt von Textdateien auf dem Terminal an (q zum beenden)
`cp`	Kopiert Dateien oder Verzeichnisse
`mv`	Verschiebt Dateien oder Verzeichnisse Wird auch zum Umbenennen verwendet
`rm`	Löscht Dateien
`cat`	Gibt den Dateieinhalt im Terminal aus
`zip` / `unzip`	Packt bzw. entpackt ZIP-Archive
`sleep`	Wartet einen angegebenen Zeitraum

- Es gibt *keine* vollständige Liste aller Befehle.
- Oft muss man viel lesen um einen passenden Befehl zu finden.

Dateien und Verzeichnisse

- Hierarchische Struktur
- Zu jeder Zeit gibt es ein aktuelles Verzeichnis: *working directory*
- Pfade sind absolut (beginnen mit '/') oder relativ (beginnen *nicht* mit '/')
- Relative Pfade werden vom momentanen working directory aus interpretiert, absolute immer vom root-Verzeichnis.
- Jeder User hat ein *home directory*
  - Nach dem Einloggen befindet man sich im *home directory*
  - Dort kann man schreiben und lesen wie man möchte

Pfade eingeben

Besondere Einträge

- `.` Aktuelles Verzeichnis
- `..` Ein Verzeichnis weiter oben im Baum ("Parent-directory")

Wildcards

- `?` Steht für *ein* beliebiges Zeichen
  `h?llo` passt auf `hallo` oder `hello`, nicht aber auf `halllo`
- `*` Steht für beliebig viele beliebige Zeichen
  `*.txt` passt auf alle Dateinamen, die auf `.txt` enden.
  `a*` passt auf alle Dateinamen, die mit `a` beginnen.

Pfade eingeben

Besondere Einträge

`.` Aktuelles Verzeichnis

`..` Ein Verzeichnis weiter oben im Baum ("Parent-directory")

Wildcards

? Steht für *ein* beliebiges Zeichen
`h?llo` passt auf `hallo` oder `hello`, nicht aber auf `halllo`

* Steht für beliebig viele beliebige Zeichen
`*.txt` passt auf alle Dateinamen, die auf `.txt` enden.
`a*` passt auf alle Dateinamen, die mit `a` beginnen.

Tilde-Expansion

- ~ wird automatisch durch die $HOME Variable ersetzt
- Funktion der Shell, nicht des Dateisystems
- Aufgerufene Programme bekommen von diesem Prozess nichts mit:

```
$ echo ~/test.txt
/home/mitarb/steen/test.txt
```

- Pfade mit ~ werden nicht in jeder Umgebung automatisch expandiert. Beispiel Python:

```
os.listdir("~") # Keine Expansion
os.listdir(os.path.expanduser("~")) # Expansion
```

Linux I

Linux I

Prozesse und Programme

Programm Anweisungen, die nacheinander ausgeführt werden. In Dateien gespeichert; kompiliert oder interpretiert.

Prozess Ein gestartetes Programm läuft in einem eigenen Prozess. Prozess läuft in einem „Kontext": Rechte, Working Directory, Umgebungsvariablen, Dateideskriptoren, ...

Subprozesse

- Alle Prozesse auf einem Computer sind hierarchisch angeordnet.
- Wird ein neuer Prozess gestartet, läuft er als Subprozess des Prozesses, von dem er gestartet wurde.
- Der Kontext des Elternprozesses wird übernommen.

Prozesse

Laufende Prozesse

- `ps` ohne Parameter zeigt laufende Prozesse, die vom momentanen Benutzer im laufenden Terminal gestartet wurden.

Beispiel (`:~$ ps -jH`)

```
  PID  PGID   SID TTY          TIME CMD
10185 10185 10185 pts/0    00:00:01 bash
10387 10387 10185 pts/0    00:00:00   ps
```

bash Die Shell selbst

ps Das ps-Kommando

Option `-e` zeigt alle laufenden Prozesse (every)

Interaktive Alternative:

```
:~$ top
```

Prozesse

Laufende Prozesse

- `ps` ohne Parameter zeigt laufende Prozesse, die vom momentanen Benutzer im laufenden Terminal gestartet wurden.

Beispiel (`:~$ ps -jH`)

```
  PID  PGID   SID TTY          TIME CMD
10185 10185 10185 pts/0    00:00:01 bash
10387 10387 10185 pts/0    00:00:00   ps
```

bash Die Shell selbst

ps Das ps-Kommando

Option `-e` zeigt alle laufenden Prozesse (every)

Interaktive Alternative:

```
:~$ top
```

Prozesse

Laufende Prozesse

- `ps` ohne Parameter zeigt laufende Prozesse, die vom momentanen Benutzer im laufenden Terminal gestartet wurden.

Beispiel (`:~$ ps -jH`)

```
  PID  PGID   SID TTY          TIME CMD
10185 10185 10185 pts/0    00:00:01 bash
10387 10387 10185 pts/0    00:00:00   ps
```

bash Die Shell selbst

ps Das ps-Kommando

Option `-e` zeigt alle laufenden Prozesse (every)

Interaktive Alternative:

```
:~$ top
```

Subprozesse erzeugen

Beispiel (`:~$ sleep 5m`)

`sleep` startet in einem Subprozess, der den gleichen Kontext hat wie die Shell.
Solange **`sleep`** läuft, wartet die Shell.

Beispiel (`:~$ sleep 5m &`)

Ein & am Ende des Kommandos schiebt den Prozess in den Hintergrund.
Die Shell wartet dann nicht, bis der Prozess beendet ist.

Subprozesse erzeugen

Beispiel (`:~$ sleep 5m`)

`sleep` startet in einem Subprozess, der den gleichen Kontext hat wie die Shell.
Solange **`sleep`** läuft, wartet die Shell.

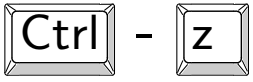
Beispiel (`:~$ sleep 5m &`)

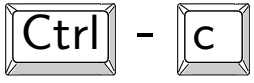
Ein & am Ende des Kommandos schiebt den Prozess in den Hintergrund.
Die Shell wartet dann nicht, bis der Prozess beendet ist.

Subprozesse managen

- Das Kommando `jobs` zeigt eine Liste aller Prozesse im Hintergrund an
- Mit `fg` und `bg` lassen sich Prozesse in den Hintergrund schieben oder in den Vordergrund holen
- Ein laufender Prozess kann mit Ctrl - z pausiert und mit Ctrl - c abgebrochen werden
- Achtung: damit ein pausierter Prozess weiterläuft, muss er mit `bg` in den Hintergrund geschoben werden

Subprozesse managen

- Das Kommando `jobs` zeigt eine Liste aller Prozesse im Hintergrund an
- Mit `fg` und `bg` lassen sich Prozesse in den Hintergrund schieben oder in den Vordergrund holen
- Ein laufender Prozess kann mit 

pausiert und mit 

abgebrochen werden
- Achtung: damit ein pausierter Prozess weiterläuft, muss er mit `bg` in den Hintergrund geschoben werden

Remote arbeiten: SSH

- **`ssh`** stellt eine *sichere* Verbindung zu einem anderen Computer her
- Argument: **`hostname`**
- Optionen:
  - **`-l <user>`** Der Username auf dem anderen Computer
  - **`-X`** Erlaubt das Benutzen graphischer Programme
- Ist der Username auf beiden Computern gleich, kann er weggelassen werden (das ist am ICL der Fall!)

Beispiele

- **`$ ssh -l steen ella`**
- **`$ ssh steen@ella`**
- **`$ ssh ella`**

Remote arbeiten

- Prozesse werden beendet, wenn die SSH-Verbindung abbricht (z.B. durch Schließen des Terminalfensters)
- Um Prozesse weiterlaufen zu lassen (etwa: die ganze Nacht / über das Wochenende), muss man Prozesse *vorher* richtig starten, zum Beispiel mit **`screen`**

screen

- Erlaubt *persistente* Sitzungen
- Zwischenschicht zwischen Terminal und Rechner

Beispiel

```
$ ssh ella
ella$ screen
ella$ # do something
ella$ C-a d
[detached]
ella$ # do something else
ella$ screen -r
```

Es gibt Alternativen zu `screen`, z.B. `tmux` (`http://tmux.sourceforge.net/`)

screen

- Erlaubt *persistente* Sitzungen
- Zwischenschicht zwischen Terminal und Rechner

Beispiel

```
$ ssh ella
ella$ screen
ella$ # do something
ella$ C-a d
[detached]
ella$ # do something else
ella$ screen -r
```

Es gibt Alternativen zu `screen`, z.B. `tmux` (`http://tmux.sourceforge.net/`)

screen

- Erlaubt *persistente* Sitzungen
- Zwischenschicht zwischen Terminal und Rechner

Beispiel

```
$ ssh ella
ella$ screen
ella$ # do something
ella$ C-a d
[detached]
ella$ # do something else
ella$ screen -r
```

Es gibt Alternativen zu `screen`, z.B. `tmux` (`http://tmux.sourceforge.net/`)

Public key authentication

- Alternative zum Passwort: Public key authentication
  - Kein ständiges Eintippen der Passwörter (z.B. bei Skripten)
  - Sehr sichere Authentifizierungsvariante
- Basiert auf Key-Pair

  Public Key wird auf dem Server abgelegt

  Private Key Verbleibt beim Nutzer. Kenntnis des privaten Schlüssels erlaubt Anmeldung am Server.

Key-Pair erstellen

1. `ssh-keygen -t rsa -b 4096 -C ''steen@cl.uni-heidelberg.de''` erzeugt Key-Pair.
   - Das Script stellt bei der Generierung einige Fragen
   - Die Defaulteinstellungen sind normalerweise in Ordnung
   - Eine Passphrase verschlüsselt den Schlüssel und verhindert Diebstahl
2. `ssh-copy-id -i  /.ssh/id_rsa steen@ella` kopiert öffentlichen Schlüssel auf ella.

Linux I

Ein- und Ausgabe

- Stream/Strom: Ein Kanal, in dem zeichenweise gelesen und geschrieben wird
- Drei Streams werden automatisch für jeden Prozess geöffnet:

  `STDOUT` Standardausgabe-Kanal – im Normalfall das Terminal

  `STDIN` Standardeingabe-Kanal, von dem der Prozess lesen kann – im Normalfall sind das Tastatureingaben

  `STDERR` Standard-Fehlerausgabe – Fehlermeldungen können entweder angezeigt oder gespeichert werden (oder beides)

Standardstreams

Beispiel (Terminal)

`STDOUT` Das Fenster am Bildschirm

`STDERR` Das Fenster am Bildschirm, allerdings ungepuffert

`STDIN` Die Tastatur

Prozesse, die im Terminal gestartet werden, „erben" diese Eigenschaften, wenn sie nicht explizit umgeleitet werden

Streams umleiten I

Ausgabe umleiten

Mit dem Zeichen $>$ am Ende eines Kommandos kann man die (Standard-)Ausgabe des Kommandos in eine Datei umleiten

Beispiel (`:~$ ps > processes.txt`)

Keine Ausgabe mehr im Terminal, stattdessen stehen die Informationen jetzt in der Datei **`processes.txt`**, wovon man sich mit **`less`** überzeugen kann.

Streams umleiten II

Eingabe umleiten

Mit dem Zeichen $<$ am Ende eines Kommandos wird die Standardeingabe des Kommandos aus einer Datei gefüllt

Beispiel (`:~$ sort < processes.txt`)

```
  26331   pts/9    00:00:00   bash
  31920   pts/9    00:00:00   ps
    PID     TTY        TIME   CMD
```

Streams umleiten III

Umleitungen

`>`	`STDOUT` umleiten, alte Datei wird gelöscht
`>>`	`STDOUT` umleiten, an alte Datei wird angehängt
`<`	`STDIN` umleiten
`2>`	`STDERR` umleiten
`&>`	`STDOUT` und `STDERR` umleiten
`2>&1`	`STDERR` an `STDOUT` senden

Pipe

Beispiel

```
:~$ ps > processes.txt
```

```
:~$ sort < processes.txt
```

Beispiel

```
:~$ ps | sort
```

Pipe

- Pipe | kombiniert verschiedene Befehle
- Prozess 2 bekommt STDOUT von Prozess 1 als STDIN

Pipe

Beispiel

```
:~$ ps > processes.txt
```

```
:~$ sort < processes.txt
```

Beispiel

```
:~$ ps | sort
```

Pipe

- Pipe | kombiniert verschiedene Befehle
- Prozess 2 bekommt STDOUT von Prozess 1 als STDIN

Linux I

Escaping

" und '

- Zeichenketten (*Strings*) werden mit Anführungszeichen " oder Apostrophen ' begrenzt
- Um Anführungszeichen oder Apostrophen als Zeichen zu verwenden, müssen sie *escaped* werden
- Escaping wird in Linux/Unix mit einem Backslash \ gemacht

Beispiele (Escaping)

- "My sister's friend's investments" ✓
- 'My sister\'s friend\'s investments' ✓
- 'My sister's friend's investments' X

Escaping

" und '

- Zeichenketten (*Strings*) werden mit Anführungszeichen " oder Apostrophen ' begrenzt
- Um Anführungszeichen oder Apostrophen als Zeichen zu verwenden, müssen sie *escaped* werden
- Escaping wird in Linux/Unix mit einem Backslash \ gemacht

Beispiele (Escaping)

- "My sister's friend's investments" ✓
- 'My sister\'s friend\'s investments' ✓
- 'My sister's friend's investments' X

Escaping

" und '

- Zeichenketten (*Strings*) werden mit Anführungszeichen " oder Apostrophen ' begrenzt
- Um Anführungszeichen oder Apostrophen als Zeichen zu verwenden, müssen sie *escaped* werden
- Escaping wird in Linux/Unix mit einem Backslash \ gemacht

Beispiele (Escaping)

- "My sister's friend's investments" ✓
- 'My sister\'s friend\'s investments' ✓
- 'My sister's friend's investments' X

Sonderzeichen

Nicht-druckbare Zeichen

- Bestimmte Zeichen des normalen Zeichensatzes sind *nicht druckbar*.
- Dazu gehören zum Beispiel Zeilenumbrüche, Wagenrückläufe, Tabulatoren oder die Backspace-Taste.
- Mit diesen Zeichen kann dennoch normal gearbeitet werden – Escaping macht es möglich.
- Für jedes dieser Steuerzeichen ist ein Escape Code der Form \<zeichen> definiert

Escape Codes

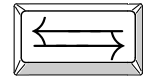
Escape Code	Name	Tastatursymbol
\b	Backspace	
\c	Control	
\f	Form feed	
\n	New line, return	
\r	Carriage return	
\t	Tab	
\\	Backslash	

Tabelle: Ausgewählte Escape Codes

Zeilenumbrüche

DOS/Windows	Carriage Return, New line	\r \n
Unix (Linux, Mac OS X)	New line	\n
Mac OS	Carriage Return	\r

Tabelle: Zeilenende nach Betriebssystem

Umwandlung

`dos2unix` , **`unix2dos`** , **`tr`** , **`sed`**

Editoren

- Text-Editoren unterstützen manuelle Texteingabe
- In unserem Kontext: Dokumentation, Quelltext, Daten, ...
- Zwei große Editoren: `Emacs` und `vi`
- Sie sollten sich mit einem von beiden beschäftigen.

Features

- Syntax-Highlighting für *alle* Programmiersprachen/Formate
- Suchfunktion (reguläre Ausdrücke!), Suchen und Ersetzen
- Copy-Cut-Paste, Undo
- Universelle Verfügbarkeit

Editoren

- Text-Editoren unterstützen manuelle Texteingabe
- In unserem Kontext: Dokumentation, Quelltext, Daten, ...
- Zwei große Editoren: **`Emacs`** und **`vi`**
- Sie sollten sich mit einem von beiden beschäftigen.

Features

- Syntax-Highlighting für *alle* Programmiersprachen/Formate
- Suchfunktion (reguläre Ausdrücke!), Suchen und Ersetzen
- Copy-Cut-Paste, Undo
- Universelle Verfügbarkeit

Linux I

Reguläre Ausdrücke

- Beschreiben eine Menge von Zeichenfolgen
- Ist eine gegebene Zeichenfolge Teil dieser Menge, *matcht* der Ausdruck auf die Zeichenfolge.

Anwendungsbeispiele

- Entfernen von Zeichen und Zeichenketten aus Texten
- Umwandeln von Zeichen in andere Zeichen
- Herausfinden, ob Zeichenketten ein bestimmtes Format haben

Reguläre Ausdrücke

- Beschreiben eine Menge von Zeichenfolgen
- Ist eine gegebene Zeichenfolge Teil dieser Menge, *matcht* der Ausdruck auf die Zeichenfolge.

Anwendungsbeispiele

- Entfernen von Zeichen und Zeichenketten aus Texten
- Umwandeln von Zeichen in andere Zeichen
- Herausfinden, ob Zeichenketten ein bestimmtes Format haben

Varianten

- Keine einheitliche Syntax für reguläre Ausdrücke
- Insbesondere Kommandozeilen-Tools nutzen oft verschiedene Varianten
- Wir schauen uns *POSIX* genauer an, da viele Kommandozeilentools POSIX oder eine Abwandlung von POSIX verwenden

Linux I

Basic Regular Expressions

- Zeichen matchen auf sich selbst
- `.` matcht ein (!) beliebiges Zeichen
- `^` matcht den Anfang der Zeichenkette oder der Zeile
- `$` matcht das Ende der Zeichenkette oder der Zeile
- `*` matcht das vorherige Zeichen 0 oder mehr Male
- `\{m, n\}` matcht das vorherige Zeichen mindestens m und maximal n mal

BRE – Bracket Expression

- `[ ]` matcht jedes einzelne Zeichen, das zwischen den Klammern steht (bracket expression)
- `[^ ]` matcht jedes einzelne Zeichen, das *nicht* zwischen den Klammern steht

Beispiele

- Einzelne Zeichen: `[aeiou]`
- Nicht einzelne Zeichen: `[^a-d]`
- Bereiche: `[a-d0-3]`
- Einfacher Bindestrich am Anfang oder Ende: `[-a-z]`

BRE – Bracket Expression

- `[ ]` matcht jedes einzelne Zeichen, das zwischen den Klammern steht (bracket expression)
- `[^ ]` matcht jedes einzelne Zeichen, das *nicht* zwischen den Klammern steht

Beispiele

- Einzelne Zeichen: `[aeiou]`
- Nicht einzelne Zeichen: `[^a-d]`
- Bereiche: `[a-d0-3]`
- Einfacher Bindestrich am Anfang oder Ende: `[-a-z]`

BRE – Backreferences

- `\( \)` Ein Block für Backreferences
- `\n` matcht den n-ten Block ($1 \leq n \leq 9$)

Beispiele

- Zwei gleiche Buchstaben hintereinander: `\(.\)\1`
- Weiter entfernte Referenz: `\(.\)h\1` (aha, oho, uhu, ...)
- Maximal neun Blöcke

BRE – Backreferences

- `\( \)` Ein Block für Backreferences
- `\n` matcht den n-ten Block ($1 \leq n \leq 9$)

Beispiele

- Zwei gleiche Buchstaben hintereinander: `\(.\)\1`
- Weiter entfernte Referenz: `\(.\)h\1` (aha, oho, uhu, ...)
- Maximal neun Blöcke

BRE – Escaping

- Um ein Zeichen zu matchen, das im Ausdruck eine Bedeutung hat, muss es *escaped* werden
- Dazu benutzt man den Backslash: `\`
- Folgende Zeichen müssen escaped werden: `[`, `]`, `.`, `*`, `^`, `$`, `\`.
- Runde und geschweifte Klammern werden *nicht* escaped, da diese ja escaped verwendet werden.

Zeichenklassen

`[[:alnum:]]`	`[a-zA-Z0-9]`
`[[:alpha:]]`	`[a-zA-Z]`
`[[:digit:]]`	`[0-9]`
`[[:lower:]]`	`[a-z]`
`[[:punct:]]`	Punctuation and Symbols
`[[:space:]]`	`[\t\r\n\v\f]`
`[[:upper:]]`	`[A-Z]`
`[[:word:]]`	`[a-zA-Z0-9_]`

Tabelle: Zeichenklassen

Extended Regular Expressions

- Erweiterung zu BRE
- Meta-characters (`\(, \), \{ \}`) werden nicht mehr escaped.
  `\(` matcht jetzt "("anstatt einen Block aufzumachen
- `?` matcht das vorige Zeichen 0 oder 1 mal
- `+` matcht das vorige Zeichen 1 oder mehr male
- `a|b` matcht "a" oder "b"
- Bei vielen Kommandozeilentools werden Extended REs mit der Option `-E` aktiviert

Linux I

Matching

`grep`

- Mit **`grep`** kann man Dateien nach Vorkommen von Zeichenketten durchsuchen
- In der Standardeinstellung werden die Zeilen, in welchen die gesuchten Zeichenketten vorkommen, ausgegeben
- Die Zeichenketten können durch reguläre Ausdrücke definiert werden

Beispiel (`:~$ grep '^$' file.txt`)

Findet alle leeren Zeilen in `file.txt`

Ersetzung

`sed`

- Werkzeug, um Datenströme zu bearbeiten
- Insbesondere für Ersetzungsregeln mit regulären Ausdrücken nützlich
- Mit dem Ausdruck **`s/a/b/g`** wird jedes Vorkomen von a durch b ersetzt
- Modifikatoren können angehängt werden
- Statt / sind auch andere Zeichen möglich, z.B. `s#/ein/pfad/#/ein/anderer/pfad#g`

Beispiel (`:~$ sed 's/groß/klein/g' file.txt`)

Ersetzt jede in **`file.txt`** vorkommende Zeichenfolge „groß" durch die Zeichenfolge „klein"

Ersetzung – Beispiele

Beispiele

- `$ sed 's/[[:space:]]/\n/g' file.txt`
- `$ sed 's/\([[:space:]]\)[[:punct:]]/\1/g'`
- `$ sed 's/\([[:lower:]]\)\([[:upper:]]\)/\1 \2/g'`

Ersetzung – Beispiele

Beispiele

- `$ sed 's/[[:space:]]/\n/g' file.txt`
  Ersetzt jedes space-Zeichen durch einen Zeilenumbruch. Jedes „Wort” steht dann auf einer eigenen Zeile
- `$ sed 's/\([[:space:]]\)[[:punct:]]/\1/g'`
- `$ sed 's/\([[:lower:]]\)\([[:upper:]]\)/\1 \2/g'`

Ersetzung – Beispiele

Beispiele

- `$ sed 's/[[:space:]]/\n/g' file.txt`
- **`$ sed 's/\([[:space:]]\)[[:punct:]]/\1/g'`**
- `$ sed 's/\([[:lower:]]\)\([[:upper:]]\)/\1 \2/g'`

Ersetzung – Beispiele

Beispiele

- `$ sed 's/[[:space:]]/\n/g' file.txt`

- **`$ sed 's/\([[:space:]]\)[[:punct:]]/\1/g'`**
  Ersetzt Space gefolgt von Punctuation-Zeichen durch Space.
- `$ sed 's/\([[:lower:]]\)\([[:upper:]]\)/\1 \2/g'`

Ersetzung – Beispiele

Beispiele

- `$ sed 's/[[:space:]]/\n/g' file.txt`
- `$ sed 's/\([[:space:]]\)[[:punct:]]/\1/g'`
- **`$ sed 's/\([[:lower:]]\)\([[:upper:]]\)/\1 \2/g'`**

Ersetzung – Beispiele

Beispiele

- `$ sed 's/[[:space:]]/\n/g' file.txt`
- `$ sed 's/\([[:space:]]\)[[:punct:]]/\1/g'`
- **`$ sed 's/\([[:lower:]]\)\([[:upper:]]\)/\1 \2/g'`**
  Ersetzt CamelCase-Ausdrücke durch Camel Case

Übung 1

Linux I

Linux I

Variablen in der Shell

In der Shell können Variablen verwendet werden.

- Deklaration: Variablenname, Gleichheitszeichen, Wert (keine Leerzeichen!)
- Benutzung: Dollarzeichen, Variablenname
- Variablen sind grundsätzlich *ungetypt*!

Beispiel

```
$ VARNAME=Hallo
$ echo $VARNAME
Hallo
$ echo $VARNAME Welt!
Hallo Welt!
$ echo $VARNAME2

$ echo ${VARNAME}2
Hallo2
```

Variablen in der Shell

In der Shell können Variablen verwendet werden.

- Deklaration: Variablenname, Gleichheitszeichen, Wert (keine Leerzeichen!)
- Benutzung: Dollarzeichen, Variablenname
- Variablen sind grundsätzlich *ungetypt*!

Beispiel

```
$ VARNAME=Hallo
$ echo $VARNAME
Hallo
$ echo $VARNAME Welt!
Hallo Welt!
$ echo $VARNAME2

$ echo ${VARNAME}2
Hallo2
```

Variablen in Strings

" und ' verhalten sich unterschiedlich!

- Double quotes: Variablenreferenzen werden ersetzt
- Apostroph/Single quote: Variablenreferenzen werden nicht ersetzt

Beispiele

```
$ TEST="A B"
$ echo "$TEST"
A B
$ echo '$TEST'
$TEST
$ TEST="A     B"
#Zeichenkette wird an Whitespace gesplittet
$ echo $TEST
A B
$ echo "$TEST"
A     B
```

Backticks

- Kommandos innerhalb von Backticks werden ausgeführt
- Standardausgabe wird als Zeichenkette zurückgegeben
- Alternative: `$( ...)`

Beispiele

```
# Variable TEST enthält Ausgabe von ls -la
$ TEST=`ls -la`
$ TEST=$( ls -la )
# Inhalt von Variable TEST wird ausgegeben
$ echo $TEST
... Ausgabe von ls -la ...
```

Arithmetik

- Operatoren +,-,*,/,**,%,
- Einfache Befehle ausführen:

```
:~$ expr 1 + 1
2
```

- Ergebnis eines arithmetischen Ausdrucks in eine Variable schreiben:

```
$ VAR1=$((1+1))
$ ((VAR2 = 1 + 1)) # Spaces optional
$ let "VAR3=1+1" # Spaces optional
$ VAR4=$((VAR3**2))
```

- In (( )) müssen Variablen nicht mit $ referenziert werden.
- Floating Point Arithmetik ist auf der Shell nicht möglich.

Umgebungsvariablen

- Werden vom Betriebssystem beim Einloggen belegt
- Können auch geändert werden,
  allerdings sollte man wissen, was man tut

`$HOME`	Der Pfad zum Homeverzeichnis
`$PWD`	Working Directory
`$OLDPWD`	Previous Working Directory
`$_`	Letztes Argument des letzten Kommandos
`$PATH`	Der Suchpfad für ausführbare Programme
`$TERM`	Angaben über das Terminal
`$OSTYPE`	Angaben über das Betriebssystem
`$BASH_VERSION`	Version der Shell
`$USER`	Der Name des Users
...	...

Exkurs: Suchpfad

- Der Suchpfad (`$PATH`) gibt an, in welchen Verzeichnissen ausführbare Dateien liegen
- Programme werden ohne genaue Pfadangabe gefunden
- Verzeichnisse werden durch Doppelpunkt getrennt
- Alle Programme (nicht aber build-ins) werden auf diese Weise gefunden: `ls` , `grep` , ...
- Mit `export` kann der Suchpfad geändert werden, so dass auch in anderen Verzeichnissen gesucht wird

```
:~$ export PATH=$PATH:/the/new/path
```

Umgebungsvariablen deklarieren

- Eine Variable wird zur Umgebungsvariable, indem das Schlüsselwort **`export`** vor die Deklaration gestellt wird

Beispiel

```
# VARNAME1 ist eine Umgebungsvariable
$ export VARNAME1=Hallo

# VARNAME2 ist eine normale Variable
$ VARNAME2=Welt
```

- Wozu sollte man Umgebungsvariablen selber deklarieren?

Umgebungsvariablen deklarieren

- Eine Variable wird zur Umgebungsvariable, indem das Schlüsselwort **`export`** vor die Deklaration gestellt wird

Beispiel

```
# VARNAME1 ist eine Umgebungsvariable
$ export VARNAME1=Hallo

# VARNAME2 ist eine normale Variable
$ VARNAME2=Welt
```

- Wozu sollte man Umgebungsvariablen selber deklarieren?

Subprozesse (Wiederholung)

- Alle Prozesse auf einem Computer sind hierarchisch angeordnet.
- Wird ein neuer Prozess gestartet, läuft er als Subprozess des Prozesses, von dem er gestartet wurde.
- Der Kontext des Elternprozesses wird übernommen.

Variablen in Subprozessen

- (Normale) Variablen sind in Subprozessen nicht (mehr) verfügbar

Beispiel

```
# Deklaration einer Variablen
$ VAR=hallo

# Start eines Subprozesses
$ bash

# Ausgabe des Inhaltes der Variable VAR
$ echo "VAR=$VAR"
VAR=
```

Umgebungsvariablen in Subprozessen

- Umgebungsvariablen sind in Subprozessen verfügbar.

Beispiel

```
# Deklaration der Umgebungsvariable
$ export VAR=hallo

# Start eines Subprozesses
$ bash

# Ausgabe des Inhalts der Variablen
$ echo "VAR=$VAR"
VAR=hallo
```

Linux I

Shell-Skripte

- Anstatt Befehle auf der Shell per Hand einzugeben kann man sie auch automatisch sequenziell verarbeiten lassen.
- Befehle werden in Dateien geschrieben.
- Die Dateien (sog. Shell-Skripte) werden ausgeführt wie ein Programm.
- Einrückung spielt grundsätzlich *keine* Rolle.
- Alle Konstruktionen, die wir bisher kennengelernt haben, sind in Skripten verwendbar:
  - Befehle, Programme
  - Ein/Ausgabeumleitungen, Pipes
  - Variablen und Umgebungsvariablen
  - ...

Shell-Skripte

- Anstatt Befehle auf der Shell per Hand einzugeben kann man sie auch automatisch sequenziell verarbeiten lassen.
- Befehle werden in Dateien geschrieben.
- Die Dateien (sog. Shell-Skripte) werden ausgeführt wie ein Programm.
- Einrückung spielt grundsätzlich *keine* Rolle.
- Alle Konstruktionen, die wir bisher kennengelernt haben, sind in Skripten verwendbar:
  - Befehle, Programme
  - Ein/Ausgabeumleitungen, Pipes
  - Variablen und Umgebungsvariablen
  - ...

Ausführen von Shell-Skripten

- Es gibt verschiedene Möglichkeiten, ein Shell-Skript auszuführen
  - Interpreter, Datei als Argument

    ```
    :~$ bash script.sh
    ```
  - Ausführbare Datei, script.sh in `$PATH`

    ```
    :~$ script.sh
    ```
  - Ausführbare Datei, script.sh nicht in `$PATH`

    ```
    :~$ /path/to/script.sh
    ```
  - Ausführbare Datei, script.sh nicht in `$PATH` (relativer Pfad)

    ```
    :~$ ./script.sh
    ```

    Punkt beachten!
- Dateirechte steuern, ob eine Datei ausführbar ist oder nicht

  `:~$ chmod ugo+x script.sh` macht die Datei ausführbar.

Ausführen von Shell-Skripten

- Es gibt verschiedene Möglichkeiten, ein Shell-Skript auszuführen
  - Interpreter, Datei als Argument

    ```
    :~$ bash script.sh
    ```

  - Ausführbare Datei, script.sh in `$PATH`

    ```
    :~$ script.sh
    ```

  - Ausführbare Datei, script.sh nicht in `$PATH`

    ```
    :~$ /path/to/script.sh
    ```

  - Ausführbare Datei, script.sh nicht in `$PATH` (relativer Pfad)

    ```
    :~$ ./script.sh
    ```

    Punkt beachten!

- Dateirechte steuern, ob eine Datei ausführbar ist oder nicht

  `:~$ chmod ugo+x script.sh` macht die Datei ausführbar.

Ausführen von Shell-Skripten

- Es gibt verschiedene Möglichkeiten, ein Shell-Skript auszuführen
  - Interpreter, Datei als Argument

    ```
    :~$ bash script.sh
    ```

  - Ausführbare Datei, script.sh in `$PATH`

    ```
    :~$ script.sh
    ```

  - Ausführbare Datei, script.sh nicht in `$PATH`

    ```
    :~$ /path/to/script.sh
    ```

  - Ausführbare Datei, script.sh nicht in `$PATH` (relativer Pfad)

    ```
    :~$ ./script.sh
    ```

    Punkt beachten!

- Dateirechte steuern, ob eine Datei ausführbar ist oder nicht

  `:~$ chmod ugo+x script.sh` macht die Datei ausführbar.

Ausführen von Shell-Skripten

- Es gibt verschiedene Möglichkeiten, ein Shell-Skript auszuführen
  - Interpreter, Datei als Argument

    ```
    :~$ bash script.sh
    ```
  - Ausführbare Datei, script.sh in `$PATH`

    ```
    :~$ script.sh
    ```
  - Ausführbare Datei, script.sh nicht in `$PATH`

    ```
    :~$ /path/to/script.sh
    ```
  - Ausführbare Datei, script.sh nicht in `$PATH` (relativer Pfad)

    ```
    :~$ ./script.sh
    ```

    Punkt beachten!
- Dateirechte steuern, ob eine Datei ausführbar ist oder nicht

  `:~$ chmod ugo+x script.sh` macht die Datei ausführbar.

Ausführen von Shell-Skripten

- Es gibt verschiedene Möglichkeiten, ein Shell-Skript auszuführen
  - Interpreter, Datei als Argument

    ```
    :~$ bash script.sh
    ```
  - Ausführbare Datei, script.sh in `$PATH`

    ```
    :~$ script.sh
    ```
  - Ausführbare Datei, script.sh nicht in `$PATH`

    ```
    :~$ /path/to/script.sh
    ```
  - Ausführbare Datei, script.sh nicht in `$PATH` (relativer Pfad)

    ```
    :~$ ./script.sh
    ```

    Punkt beachten!
- Dateirechte steuern, ob eine Datei ausführbar ist oder nicht

  `:~$ chmod ugo+x script.sh` macht die Datei ausführbar.

Ausführen von Shell-Skripten

- Es gibt verschiedene Möglichkeiten, ein Shell-Skript auszuführen
  - Interpreter, Datei als Argument

    ```
    :~$ bash script.sh
    ```
  - Ausführbare Datei, script.sh in `$PATH`

    ```
    :~$ script.sh
    ```
  - Ausführbare Datei, script.sh nicht in `$PATH`

    ```
    :~$ /path/to/script.sh
    ```
  - Ausführbare Datei, script.sh nicht in `$PATH` (relativer Pfad)

    ```
    :~$ ./script.sh
    ```

    Punkt beachten!
- Dateirechte steuern, ob eine Datei ausführbar ist oder nicht

  `:~$ chmod ugo+x script.sh` macht die Datei ausführbar.

Ein einfaches Shell-Skript I

- Am Anfang jedes Shell-Skripts steht die sog. Shebang: `#!/bin/bash`
- Danach folgen beliebige Kommandos: Shell-Builtins sowie Programme des Systems

Ein einfaches Shell-Skript II

Beispiel (Backup-Szenario)

- Sicherungskopie von Verzeichnis **`Documents`**
- Ziel: Verzeichnis **`/mnt/backup`**

Beispiel (backup1.sh)

```
#!/bin/bash

cp -r /home/nils/Documents /mnt/backup
```

Ein Shell-Skript mit Variablen

- Variablen und Umgebungsvariablen lassen sich einfach einbauen
- Sie werden dann *zur Laufzeit* ersetzt

Beispiel (backup2.sh)

```
#!/bin/bash

cp -r $HOME/Documents /mnt/backup/$USER
```

Positionsvariablen

- Einige Variablen ergeben nur in Verbindung mit Skripten Sinn
- Sie erlauben Zugriff auf den Orts des Skripts und die Argumente

`$0`	Dateiname des Skripts
`$1, $2, …`	Parameter, die dem Skript auf der Kommandozeile mitgegeben wurden
`$#`	Anzahl der Parameter
`$@`	Die Parameter als Liste

Ein Shell-Skript mit Positionsvariablen

Beispiel (backup3.sh)

```
#!/bin/bash

# Kopiert das als Argument angegebene Verzeichnis
# aus dem Homeverzeichnis des angemeldeten Users
cp -r $HOME/$1 /mnt/backup/$USER
```

Beispiel

```
$ backup3.sh Documents
```

Ein Shell-Skript mit Positionsvariablen

Beispiel (backup3.sh)

```
#!/bin/bash

# Kopiert das als Argument angegebene Verzeichnis
# aus dem Homeverzeichnis des angemeldeten Users
cp -r $HOME/$1 /mnt/backup/$USER
```

Beispiel

```
$ backup3.sh Documents
```

Linux I

Kontrollstrukturen

- Die Bash bietet auch Kontrollstrukturen
  - `if`
  - `while` , `until`
  - `case`
  - `for`
  - `select`
- Damit lassen sich vollwertige Programme in der Shell realisieren

Kontrollstrukturen: if

- `if` führt einen angegebenen Test aus
- Wenn der Test 0 zurückgibt, wird der Code ausgeführt
- Optional: Sonst wird anderer Code ausgeführt

Warum Null?

- Traditionellerweise ist 0 in der Unix-Welt das Zeichen für Erfolg
- Als Merkhilfe: die Zahl repräsentiert die Zahl der Fehler
- Wenn 0 Fehler aufgetreten sind, ist alles gutgegangen

Kontrollstrukturen: if

- `if` führt einen angegebenen Test aus
- Wenn der Test 0 zurückgibt, wird der Code ausgeführt
- Optional: Sonst wird anderer Code ausgeführt

Warum Null?

- Traditionellerweise ist 0 in der Unix-Welt das Zeichen für Erfolg
- Als Merkhilfe: die Zahl repräsentiert die Zahl der Fehler
- Wenn 0 Fehler aufgetreten sind, ist alles gutgegangen

Kontrollstrukturen: if und Tests

Syntax

```
if TEST
then
    COMMANDS
else
    COMMANDS
fi
```

`COMMANDS`

Beliebige Kommandos

`TEST`

- `[[ … ]]`
  Extended Test Command
- Vergleiche
  - Zeichenketten
  - Zahlen
  - Daten
  - …
- Dateitests
  - Existenz
  - Letzte Änderung
  - Datei/Verzeichnis
  - …

Kontrollstrukturen: if und Tests

Syntax

```
if TEST
then
    COMMANDS
else
    COMMANDS
fi
```

COMMANDS

Beliebige Kommandos

TEST

- `[[ … ]]`
  Extended Test Command
- Vergleiche
  - Zeichenketten
  - Zahlen
  - Daten
  - …
- Dateitests
  - Existenz
  - Letzte Änderung
  - Datei/Verzeichnis
  - …

Kontrollstrukturen: if und Tests

Syntax

```
if TEST
then
    COMMANDS
else
    COMMANDS
fi
```

COMMANDS

Beliebige Kommandos

TEST

- `[[ … ]]`
  Extended Test Command
- Vergleiche
  - Zeichenketten
  - Zahlen
  - Daten
  - …
- Dateitests
  - Existenz
  - Letzte Änderung
  - Datei/Verzeichnis
  - …

Ein Skript mit Test

Beispiel (backup4.sh)

```
#!/bin/bash

# Kopiert das als Argument angegebene Verzeichnis
# aus dem Homeverzeichnis des angemeldeten Users

# Testet, ob es existiert
if [[ -e $HOME/$1 ]]
then
        cp -r $HOME/$1 /mnt/backup/$USER
else
        # Gibt Fehlermeldung aus, wenn nicht
        echo "$HOME/$1 does not exist."
fi
```

Ein Skript mit verschalteten Tests

Beispiel (backup4b.sh)

```
#!/bin/bash

# Testet, ob Verzeichnis existiert
if [[ -e $HOME/$1 ]]
then
        # Testet, ob es ein Verzeichnis ist
        if [[ -d $HOME/$1 ]]
        then
                cp -r $HOME/$1 /mnt/backup/$USER
        else
                echo "$HOME/$1 is not a directory."
        fi
else
        # Gibt Fehlermeldung aus, wenn nicht
        echo "$HOME/$1 does not exist."
fi
```

Vergleichsoperatoren

Integer

`-eq`	is equal to
`-ne`	is not equal to
`-gt`	is greater than
`-ge`	is greater than or equal to
…	…

String

`=`	is equal to
`<`	is less than (ASCII order)
`-z`	is null (has zero length)
`-n`	is not null
…	…

Achtung!

- Single und double quotes und die Art der Klammerung des Testausdrucks ändern das Verhalten des Tests!
- `[[ … ]]`, `[ … ]`, `(( … ))`

Vergleichsoperatoren

Integer

`-eq`	is equal to
`-ne`	is not equal to
`-gt`	is greater than
`-ge`	is greater than or equal to
…	…

String

`=`	is equal to
`<`	is less than (ASCII order)
`-z`	is null (has zero length)
`-n`	is not null
…	…

Achtung!

- Single und double quotes und die Art der Klammerung des Testausdrucks ändern das Verhalten des Tests!
- `[[ … ]]`, `[ … ]`, `(( … ))`

Dateitests

`-e`	Datei existiert
`-f`	Reguläre Datei
`-s`	Dateigröße ungleich Null
`-d`	Datei ist ein Verzeichnis
`-x`	Datei ist ausführbar (für den User, der den Test ausführt)
`f1 -nt f2`	f1 ist neuer als f2
…	…

Referenz

Wo schlägt man das nach?

- Auf der man-page der Bash (`:~$ man bash`)
- Advanced Bash Scripting Guide
  `tldp.org/LDP/abs/html/index.html`
- Man-page als Webseite
  `www.gnu.org/software/bash/manual/bashref.html`

Kontrollstrukturen: for

- Eine **for** -Schleife wiederholt eine Reihe von Anweisungen
- Anzahl der Iterationen vorher festgelegt

Syntax

```
for ARG in [LIST]
do
      COMMANDS
done
```

LIST

- **LIST** ist eine Liste
- Variable **ARG** wird der Reihe nach auf die Elemente der Liste gesetzt

Kontrollstrukturen: for – Beispiel

Beispiel (backup5.sh)

```
#!/bin/bash

# Iteriere ueber alle Argumente der Kommandozeile
for arg in "$@"
do
  if [[ -e $HOME/$arg ]]
  then
    if [[ -d $HOME/$arg ]]
    then
      cp -r $HOME/$arg /mnt/backup/$USER
    else
      echo "$HOME/$arg is not a directory."
    fi
  else
    echo "$HOME/$arg does not exist."
  fi
done
```

Kontrollstrukturen: for – Sequenzen

- Schleifen mit Zählervariable

```
for i in 1 2 3 4 5
do
      COMMANDS
done
```

- Lange Sequenzen: {1..1000}

```
for i in {1..1000}
do
      COMMANDS
done
```

Kontrollstrukturen: for – Sequenzen

- Schleifen mit Zählervariable

```
for i in 1 2 3 4 5
do
      COMMANDS
done
```

- Lange Sequenzen: `{1..1000}`

```
for i in {1..1000}
do
      COMMANDS
done
```

Kontrollstrukturen: for – Dateien

- Automatisch über Dateien in einem Verzeichnis iterieren
- `ls` liefert eine Liste von Dateien in einem Verzeichnis
- `$( … )` wird benutzt, um die Ausgabe von einem Kommando als Wert zuzuweisen
- `$( ls )` liefert eine Liste von Dateien in einem Verzeichnis als Wert für eine Variable

Beispiel

```
for file in $( ls directory )
do
  echo $file
done
```

Iterieren über Dateien – Beispiel

Beispiel (backup6.sh)

```
#!/bin/bash

for arg in "$@"
do
  # Iteriere ueber alle Dateien in $arg
  for directory in $( ls $arg )
  do
    if [[ -d $arg/$directory ]]
    then
      cp -r $arg/$directory /mnt/backup/$USER
    fi
  done
done
```

Übung 2

Mittagspause

Montag: Linux II

2 Linux II
- Encoding and Locale
- AWK

Linux II

Linux II

Was ist Encoding?

- Jedem benötigten Zeichen wird eine Zahl zugeordnet
- Welches Zeichen welcher Zahl zugeordnet wird, bezeichnet man als *Zeichencode*
- Das Format, in dem der Zeichencode (im Computer) repräsentiert wird, heißt *Encoding*

Zahlendarstellung im Computer

- Computer arbeiten binär (= zur Basis 2)
- Speicher: | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
- Ein Bit: | 1 | oder | 0 |
- Acht Bit werden ein Byte genannt und können 256 verschiedene Zustände repräsentieren (z.B. Zahlen zwischen 0 und 255)

1	1	1	0	0	0	1	1

1 Byte

Einschränkungen

- Wieviel Platz ein Zeichen bekommt, entscheidet darüber, wieviele Zeichen man unterscheiden kann
- Ist jedes Zeichen z.B. 4 Bit lang, können 16 Zeichen unterschieden werden
- Alle Zeichen sollten gleich viel Platz belegen; ggf. wird mit Nullen aufgefüllt

Linux II

ASCII

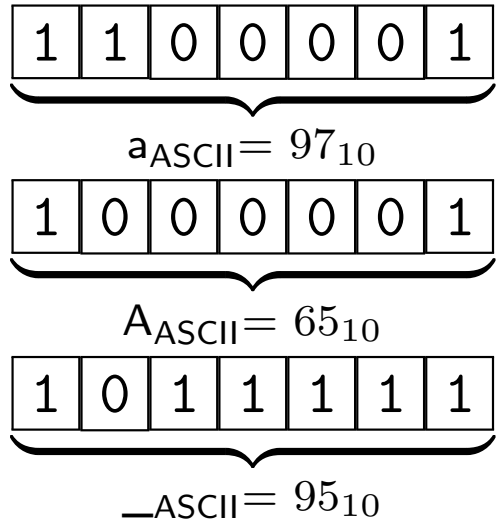
American Standard Code for Information Interchange

- Erstmals publiziert 1963
- Definiert $2^7 = 128$ Codepunkte
- ASCII enthält 33 nicht-druckbare und 95 druckbare Zeichen
- Alle Codepunkte können in 1 Byte gespeichert werden.
- Das 8. Bit dient als Kontrollsignal

Druckbare ASCII-Zeichen

```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

Beispiele



Latin-1 / ISO 8859-1

ISO: International Organization for Standardization

- ASCII reicht nicht aus, um z.B. deutsche Umlaute darzustellen
- Erweiterung von ASCII: Latin-1
- In Latin-1 werden alle 8 Bit für 256 Zeichen genutzt
- Die meisten westeuropäischen Sprachen sind darstellbar
- Erste 7 Bit entsprechen denen in ASCII, achtes Bit sorgt dann für 127 zusätzliche Umlaute und Sonderzeichen

Beispiele

Latin-1 / ISO 8859-1

ISO: International Organization for Standardization

- ASCII reicht nicht aus, um z.B. deutsche Umlaute darzustellen
- Erweiterung von ASCII: Latin-1
- In Latin-1 werden alle 8 Bit für 256 Zeichen genutzt
- Die meisten westeuropäischen Sprachen sind darstellbar
- Erste 7 Bit entsprechen denen in ASCII, achtes Bit sorgt dann für 127 zusätzliche Umlaute und Sonderzeichen

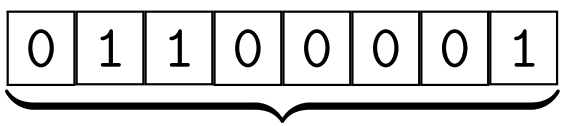
Beispiele

Latin-1 / ISO 8859-1

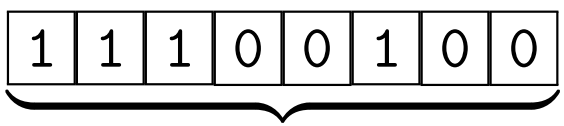
ISO: International Organization for Standardization

- ASCII reicht nicht aus, um z.B. deutsche Umlaute darzustellen
- Erweiterung von ASCII: Latin-1
- In Latin-1 werden alle 8 Bit für 256 Zeichen genutzt
- Die meisten westeuropäischen Sprachen sind darstellbar
- Erste 7 Bit entsprechen denen in ASCII, achtes Bit sorgt dann für 127 zusätzliche Umlaute und Sonderzeichen

Beispiele



$a_{\text{Latin-1}} = a_{\text{ASCII}} = 97_{10}$



$ä_{\text{Latin-1}} = 228_{10}$

Unicode

- Auch Latin-1 reicht nicht aus, um alle Zeichen darzustellen
- Im Chinesischen gibt es mehr als 47.000 verschiedene Zeichen
- 8-Bit-Zeichensätze bieten nur Raum für 256 Zeichen

Unicode

- Kodiert alle Zeichen dieser Welt
- Insgesamt 1.114.112 Codepunkte
- Verschiedene Encodings: UTF-8, UTF-16, UTF-32
- Unterschied zu Latin1: Anzahl der verwendeten Bytes kann variieren (UTF-8, UTF-16)

Unicode

- Auch Latin-1 reicht nicht aus, um alle Zeichen darzustellen
- Im Chinesischen gibt es mehr als 47.000 verschiedene Zeichen
- 8-Bit-Zeichensätze bieten nur Raum für 256 Zeichen

Unicode

- Kodiert alle Zeichen dieser Welt
- Insgesamt 1.114.112 Codepunkte
- Verschiedene Encodings: UTF-8, UTF-16, UTF-32
- Unterschied zu Latin1: Anzahl der verwendeten Bytes kann variieren (UTF-8, UTF-16)

UTF-8

- ASCII-kompatibel (ASCII-Zeichen werden so wie in ASCII kodiert)
- Nicht-ASCII-Zeichen sind 2-4 Byte lang
- Höchste Bits des ersten Bytes zeigen an, wie lang das Zeichen ist
  - $\boxed{0} \rightarrow$ 1 Byte
  - $\boxed{1}\boxed{1}\boxed{0} \rightarrow$ 2 Byte
  - $\boxed{1}\boxed{1}\boxed{1}\boxed{0} \rightarrow$ 3 Byte
  - $\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{0} \rightarrow$ 4 Byte
- Darauffolgende Bytes haben alle MSB auf 1 und das nachfolgende auf 0x (MSB: most significant bit, LSB: least significant bit)

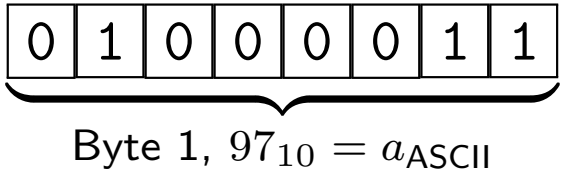
UTF-8 – Beispiele

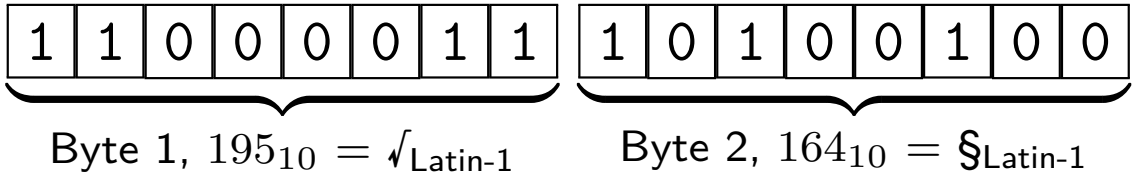
Beispiele

- $\underbrace{\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}}_{\text{Byte 1, } 97_{10} = a_{\text{ASCII}}} = \text{a}_{\text{UTF-8}}$
- $\underbrace{\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}}_{\text{Byte 1, } 195_{10} = \sqrt{}_{\text{Latin-1}}}\underbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}}_{\text{Byte 2, } 164_{10} = \S_{\text{Latin-1}}} = \text{ä}_{\text{UTF-8}}$
- $\underbrace{\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}}_{\text{Byte 1, } 224_{10} = \text{à}_{\text{Latin-1}}}\underbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{0}}_{\text{Byte 2, } 188_{10} = ¼_{\text{Latin-1}}}\underbrace{\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}}_{\text{Byte 3, } 129_{10}}$

UTF-8 – Beispiele

Beispiele

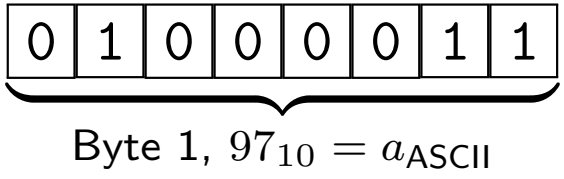
- 

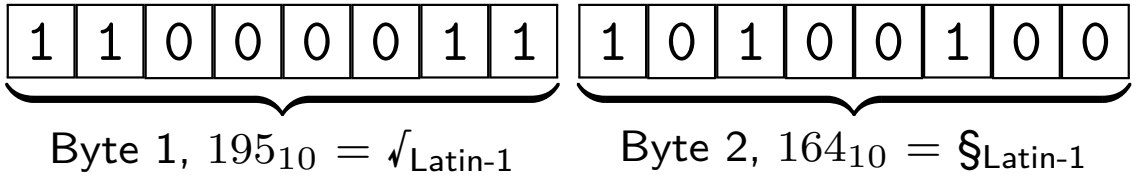
$= \text{a}_{\text{UTF-8}}$
- 

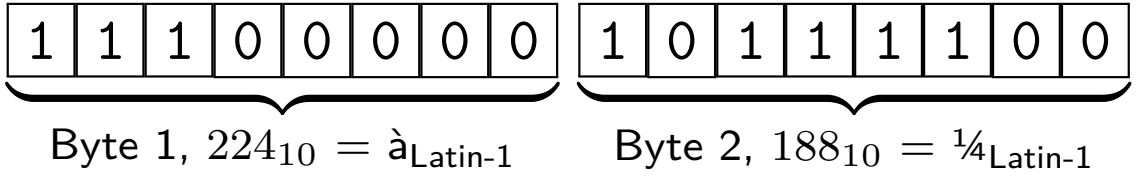
$= \text{ä}_{\text{UTF-8}}$
-

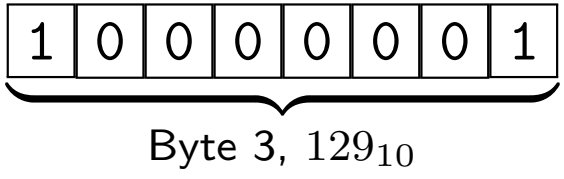
UTF-8 – Beispiele

Beispiele

- 

$= \text{a}_{\text{UTF-8}}$
- 

$= \ddot{\text{a}}_{\text{UTF-8}}$
- 



Encoding erkennen

- Encoding erkennen ist schwer!
- Validierung von Bitfolgen als gültige oder ungültige Zeichen ist erst mit Unicode wieder möglich
- Viele Encodings haben Varianten, die in einzelnen Zeichen abweichen

Hex	Dez	8859-1	8859-2	8859-8	MacRoman
BE_{16}	190	¾	ž	¾	æ
CF_{16}	207	Ï	Ď		œ
$E4_{16}$	228	ä	ä		‰

Tabelle: Unterschiede zwischen Latin-1, Latin-2, Hebrew und MacRoman

Encoding erkennen

- Encoding erkennen ist schwer!
- Validierung von Bitfolgen als gültige oder ungültige Zeichen ist erst mit Unicode wieder möglich
- Viele Encodings haben Varianten, die in einzelnen Zeichen abweichen

Hex	Dez	8859-1	8859-2	8859-8	MacRoman
BE_{16}	190	¾	ž	¾	æ
CF_{16}	207	Ï	Ď		œ
$E4_{16}$	228	ä	ä		‰

Tabelle: Unterschiede zwischen Latin-1, Latin-2, Hebrew und MacRoman

Konvertierung

- Tools zur automatischen Konvertierung: **`recode`** , **`iconv`**
- Lesen *nicht* von der Standardeingabe
- Ein- und Ausgabe direkt über Dateien

Umwandlung aller Dateien im Verzeichnis (Latin-1 → UTF-8)

```
for file in $( ls directory )
    do
    recode l1..u8 $file
done
```

Linux II

Spracheinstellungen

Nicht nur Encoding – Nicht nur Sprache

- Sprache von Fehlermeldungen, Hilfetexte
- Default-Papiergröße für's drucken (DIN A4 oder US-Letter)
- Währungszeichen (€, $, ¥...)
- Dezimal- und Tausendertrennzeichen (1,000.00 vs. 1.000,00)
- Formatierung von Telefonnummern, Adressen, ...

Locale

- `locale` zeigt die aktuellen Einstellungen an

Spracheinstellungen

Nicht nur Encoding – Nicht nur Sprache

- Sprache von Fehlermeldungen, Hilfetexte
- Default-Papiergröße für's drucken (DIN A4 oder US-Letter)
- Währungszeichen (€, $, ¥...)
- Dezimal- und Tausendertrennzeichen (1,000.00 vs. 1.000,00)
- Formatierung von Telefonnummern, Adressen, ...

Locale

- `locale` zeigt die aktuellen Einstellungen an

Spracheinstellungen in Umgebungsvariablen

- Spracheinstellungen werden in Umgebungsvariablen gespeichert

`LC_ALL`	Alle Einstellungen
`LC_CTYPE`	Zeichenklassifikation, Groß- und Kleinschreibung
`LC_COLLATE`	Sortierung
`LC_TIME`	Zeit und Datumsformat
`LC_NUMERIC`	Zahlenangaben
`LC_MONETARY`	Zahlenangaben mit Währung
`LC_MESSAGES`	Fragen, Fehlermeldungen, Dialoge
`LC_PAPER`	Papiergröße
`LC_NAME`	Namenformatierung
`LC_ADDRESS`	Adress- und Ortsangaben
`LC_TELEPHONE`	Telefonnummern
`LC_MEASUREMENT`	Maßeinheiten (Metric, Imperial, ...)

Zeichensatz und Schriftart

- Es reicht nicht, die Umgebungsvariablen zu ändern
- Die Schriftart (des Terminals) muss den Zeichensatz unterstützen
- Ändern der Schriftart:
  `Terminal` → `Set Character Encoding` / `Zeichenkodierung`

Ändern der Umgebungsvariablen

- ```
  :~$ export LC_LANG=de_DE.utf8
  ```
- ...
  ```

Linux II

AWK

- AWK ist eine Skriptsprache für strukturierte Textdaten
- GNU-Variante: GAWK
- AWK operiert auf zeilenorientierten Daten
- Jede Zeile repräsentiert ein Dokument, und jedes Dokument besteht aus mehreren Feldern.
- Die Felder sind standardmäßig durch Leerzeichen getrennt, andere Trenner sind möglich.
- Grundidee: Jede Zeile wird als ein Array aufgefasst:
  - `$0`: die komplette Zeile
  - `$1`: das erste Feld
  - ...

Grundlagen

- Ein AWK-Skript besteht aus Bedingungen und Anweisungen

```
Bedingung { Anweisungen }
```

- Wenn eine Bedingung erfüllt ist, wird der Anweisungsblock ausgeführt.
- Ist die Bedingung leer, wird die Anweisung immer ausgeführt.
- Ist die Anweisung leer, wird standardmäßig die gesamte Zeile ausgegeben, wenn die Bedingung greift.

Anweisungen

- Werden immer in `{ }` geschrieben
- Wenn keine Anweisung angegeben wird: **`{ print $0 }`**

Ausdruck	Funktionsaufruf, Wertzuweisung einer Variable (`{ VAR=0 }`)
Kontrollstruktur	**`if`**, **`for`**, ...
Ein- und Ausgabe-Statements	**`print`**, **`getline`**, ...(`{ print "/"$1 }`)
Deletion-Statements	Elemente aus Arrays löschen

`http://www.gnu.org/software/gawk/manual/html_node/Action-Overview.html`

Bedingungen

- Grundsätzlich: jeder `awk`-Ausdruck
- Wenn die Bedingung nicht den Wert 0 bzw. `""` hat, wird die Anweisung ausgeführt.
- Beispiel 1: Regulärer Ausdruck `$1 ~ /[a-c]/`
- Beispiel 2: Vergleich `$1=="word"`
- `BEGIN`: Die folgende Anweisung wird ausgeführt, *bevor* Daten gelesen werden.
- `END`: Die folgende Anweisung wird ausgeführt, *nachdem* alle Daten gelesen wurden.

`http://www.gnu.org/software/gawk/manual/html_node/Pattern-Overview.html`

Beispieldaten

```
290 india nnp
293 large jj
300 person nn
302 four cd
302 perform vbn
302 very rb
303 buddhist nnp
303 system nn
303 term nns
305 culture nn
```

Skriptaufruf

Auf der Kommandozeile

AWK-Skript muss in ' stehen:

```
:~$ gawk '$0 ~ /foo/ {print $1}' data.csv
```

In einer Datei

AWK-Skript kann aus einer Datei gelesen werden

```
:~$ gawk -f script.awk data.csv
```

Beispielskripte

material/sample1.awk

```
$3 == "nn" { print $2 }
```

Beispielskripte

material/sample1.awk

```
$3 == "nn" { print $2 }
```

material/sample2.awk

```
BEGIN { NOUNS=0 }
$3 == "nn" { NOUNS += $1 }
END { print NOUNS }
```

```
:~$ gawk -f material/sample2.awk material/data.csv
1827
```

Built-in Variables

AWK verwendet einige eingebaute Variablen, hier nur zwei wichtige Beispiele:

- FS *field separator* – Zeichen oder regulärer Ausdruck
- NR *number of records* – wieviele Zeilen bereits gelesen wurden

Field Separator ändern

```
:~$ gawk 'BEGIN{FS="\t"}{print $1}'
```

www.gnu.org/software/gawk/manual/gawk.html#Built_
002din-Variables

Weitere Beispiele

Nützliche AWK-Einzeiler

- `http://www.pement.org/awk/awk1line.txt`
- `http://www.catonmat.net/blog/awk-one-liners-explained-part-one/`

Übung 3

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>`

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

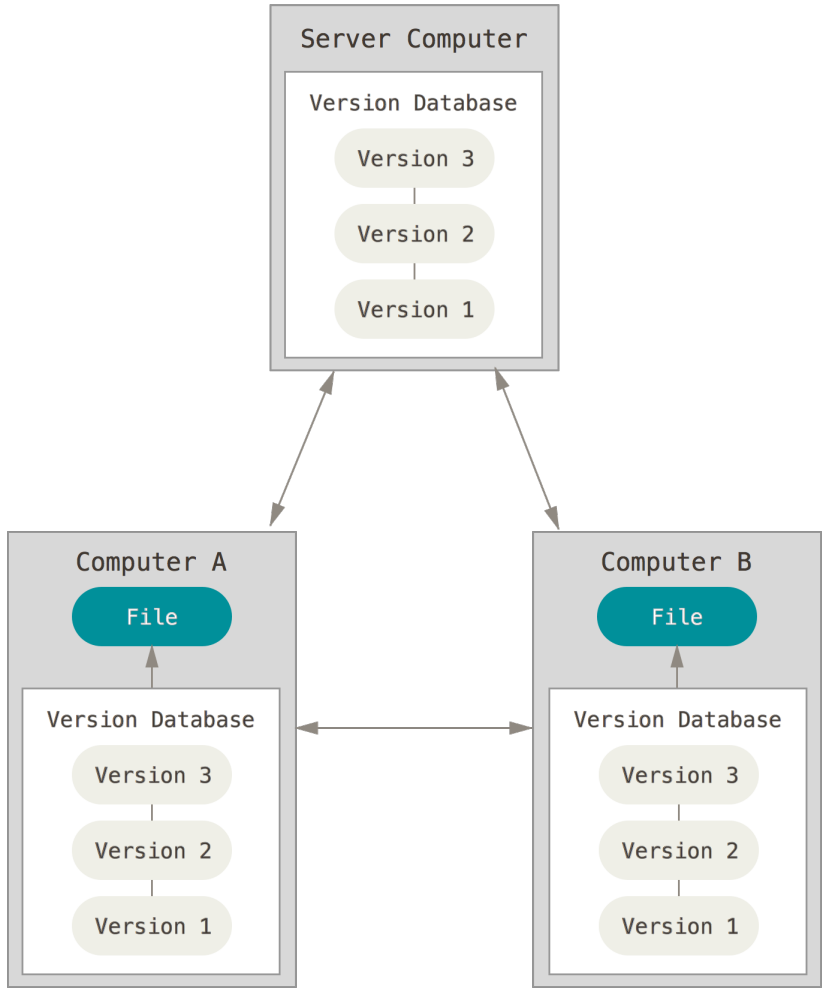
Es gibt eine Vielzahl von Versionsverwaltungssystemen:

- 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:

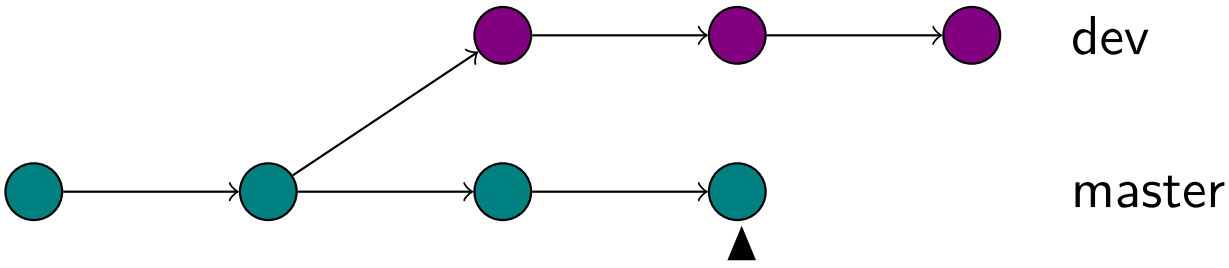
```
:~$ git init <folder>
```

- Oder man “klont” ein vorhandenes Repository:

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

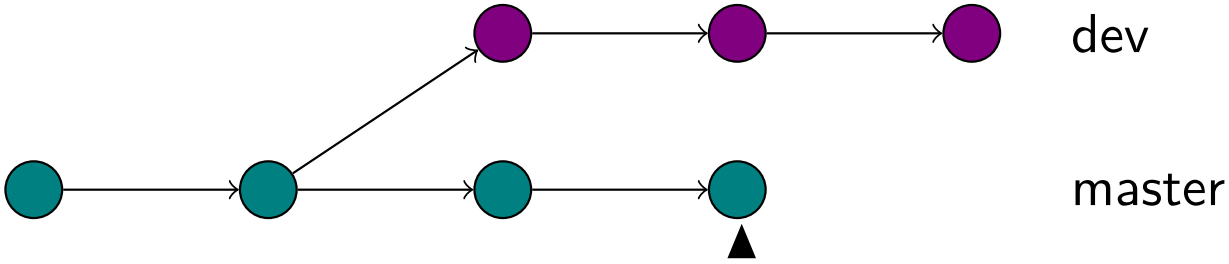
Visualisierung

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

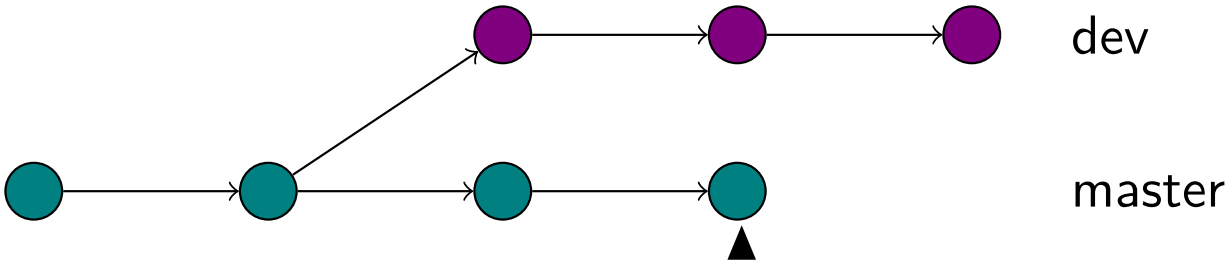


Visualisierung

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”

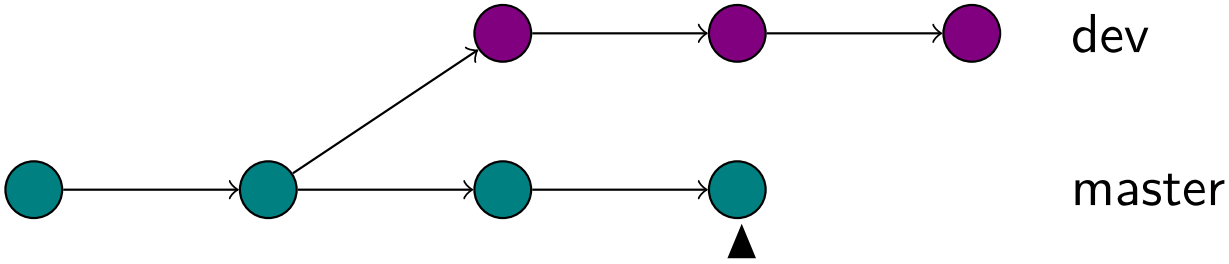
remove

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

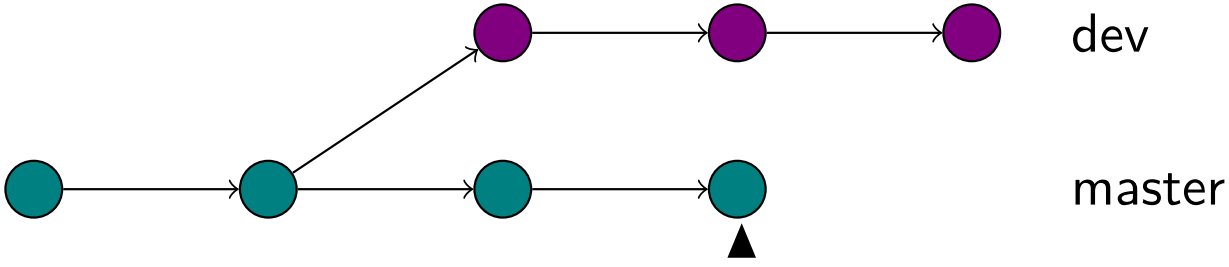
- Dateien unter Versionskontrolle dürfen **nicht** einfach gelöscht werden!
- Dateien müssen per git gelöscht werden

Visualisierung: Comitten

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

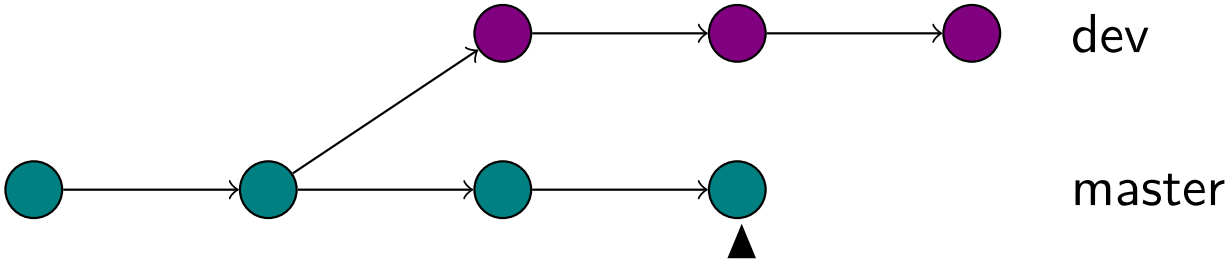


Lokal vor **`git commit`**:

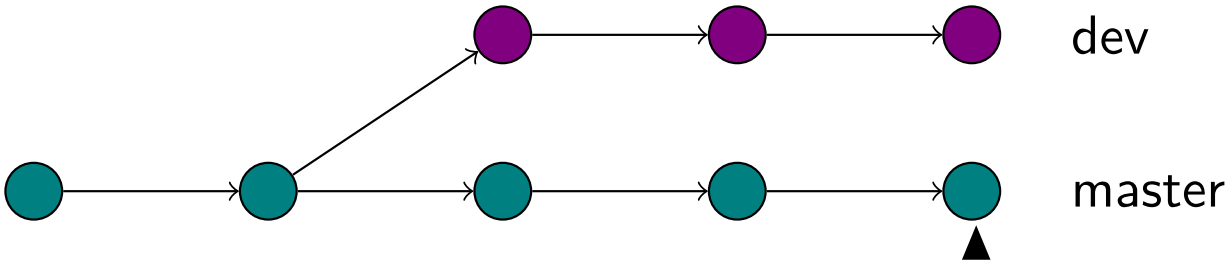


Visualisierung: Comitten

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



Lokal nach **`git commit`**:



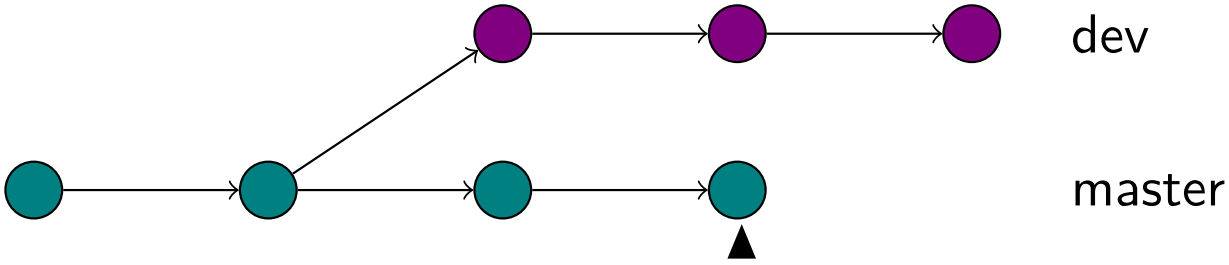
push

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

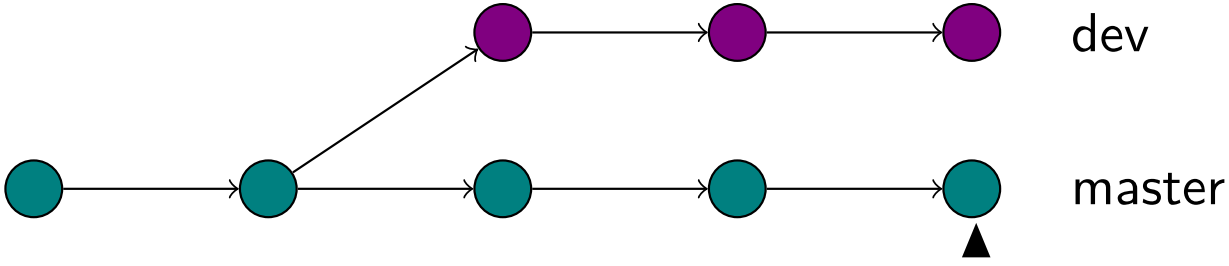
- aktualisiert das Repository names **<remote>** mit dem Status der lokalen Version des Repositorys
- default: **origin**
- Dateikonflikte sind möglich, falls jemand anderes schon etwas aktualisiert hat – dazu kommen wir später

Visualisierung: push

`origin` vor `git push origin`:

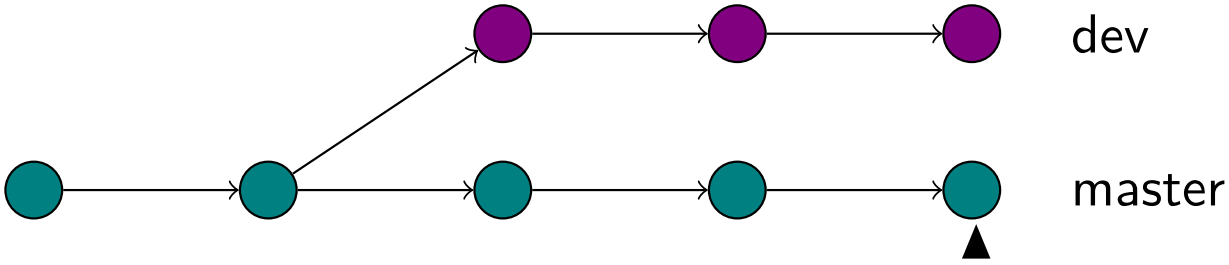


Lokal:

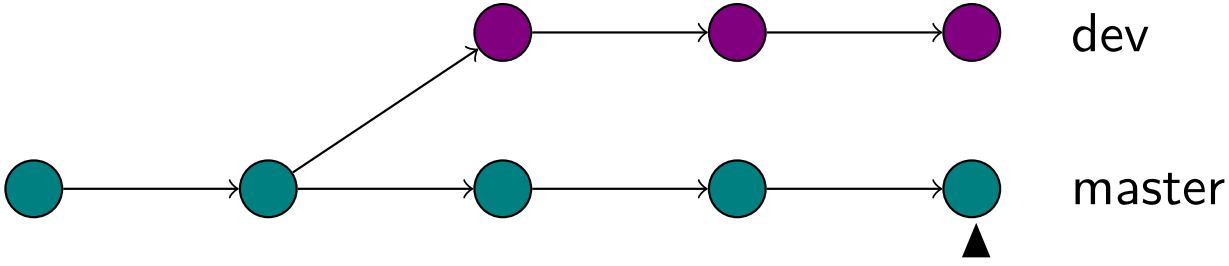


Visualisierung: push

`origin` nach `git push origin`:



Lokal:



fetch

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

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

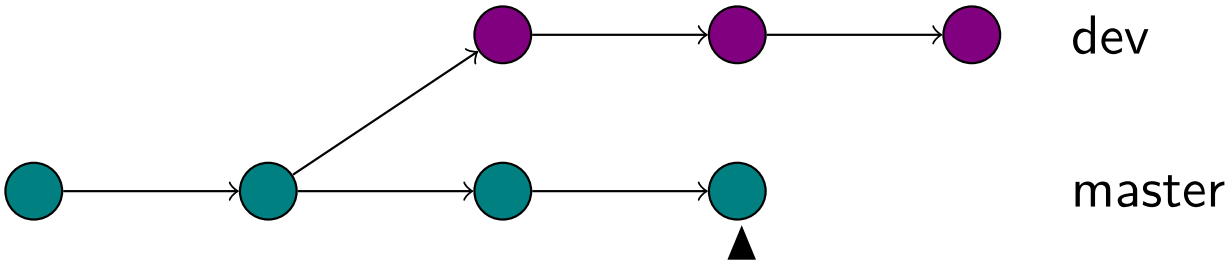
merge

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

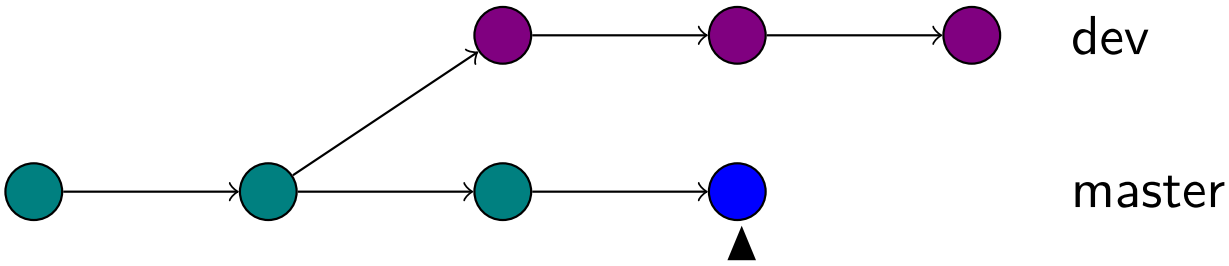
- fügt angegebenen Branch und aktuellen zusammen
- eventuell müssen Dateikonflikte aufgelöst werden
- falls keine Konflikte: es wird automatisch comittet!

Visualisierung: fetch und merge

`origin`:

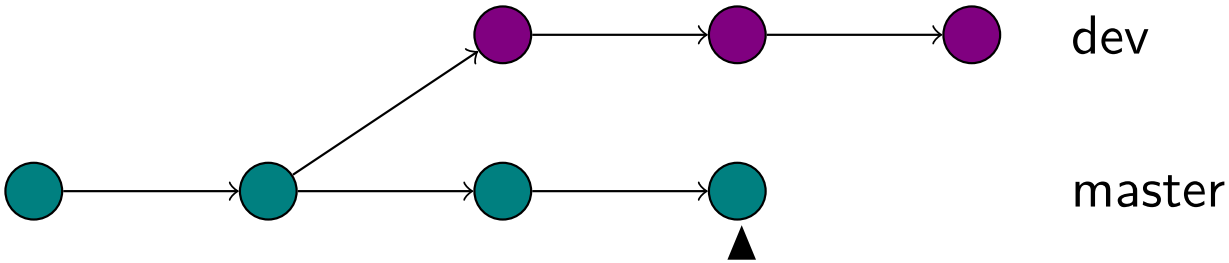


Lokal vor `git fetch origin`:

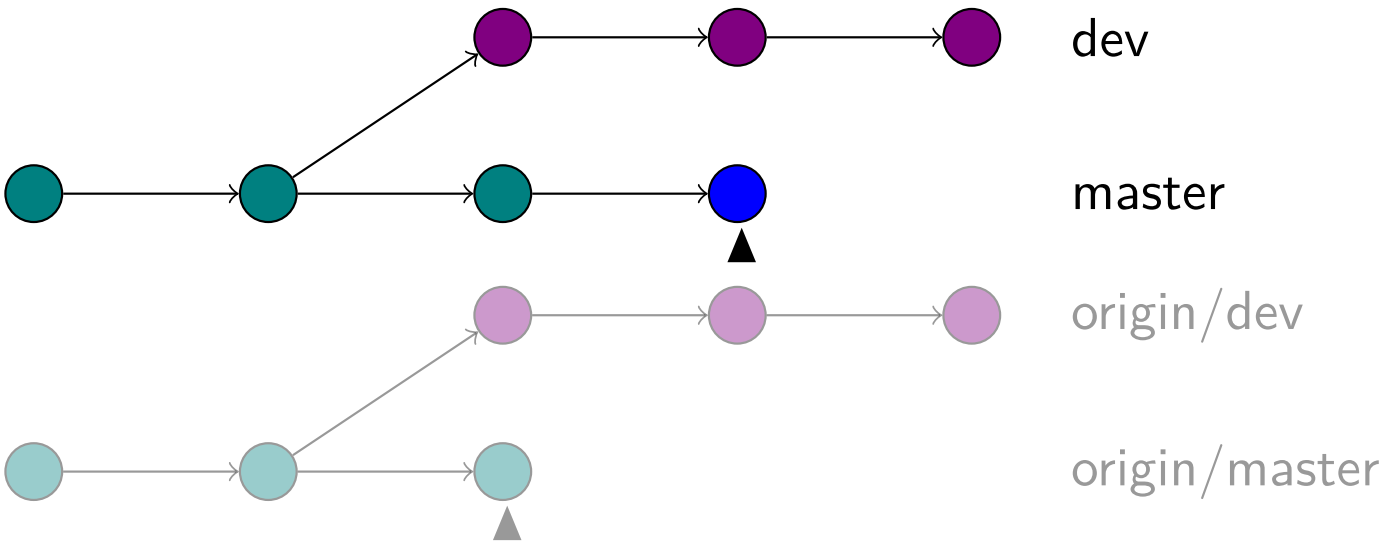


Visualisierung: fetch und merge

`origin`:

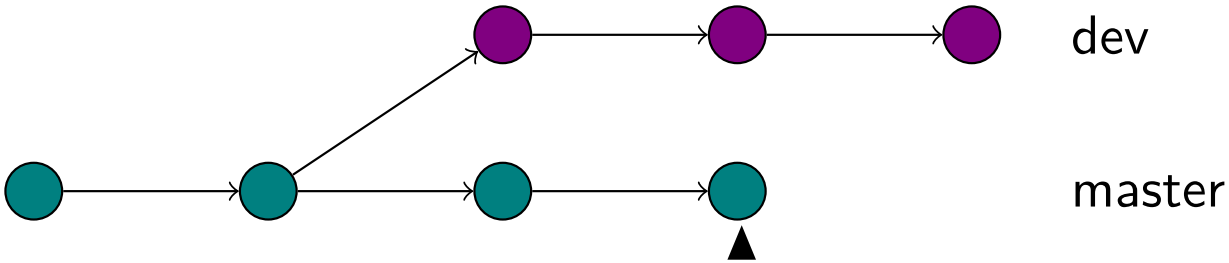


Lokal vor **`git fetch origin`**:

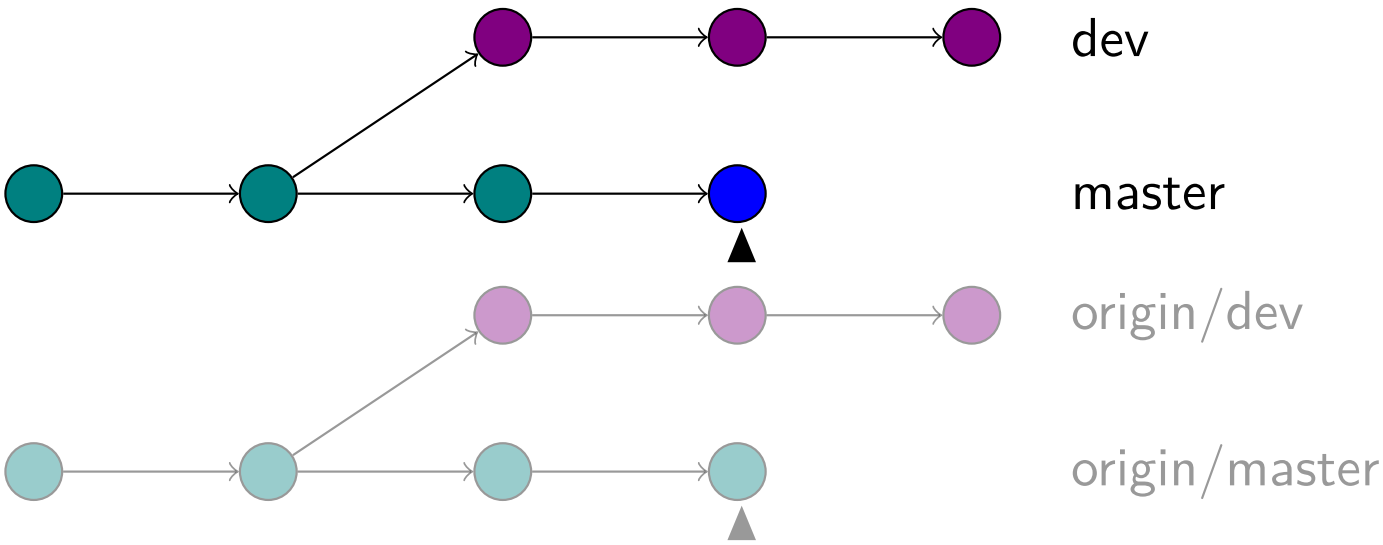


Visualisierung: fetch und merge

`origin`:

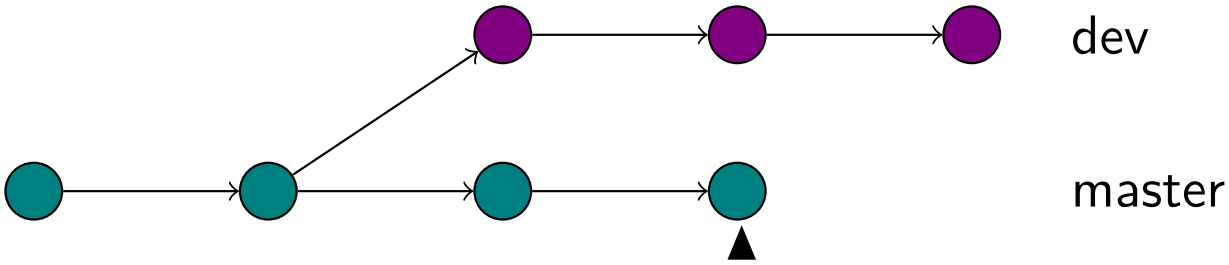


Lokal nach `git fetch origin`:

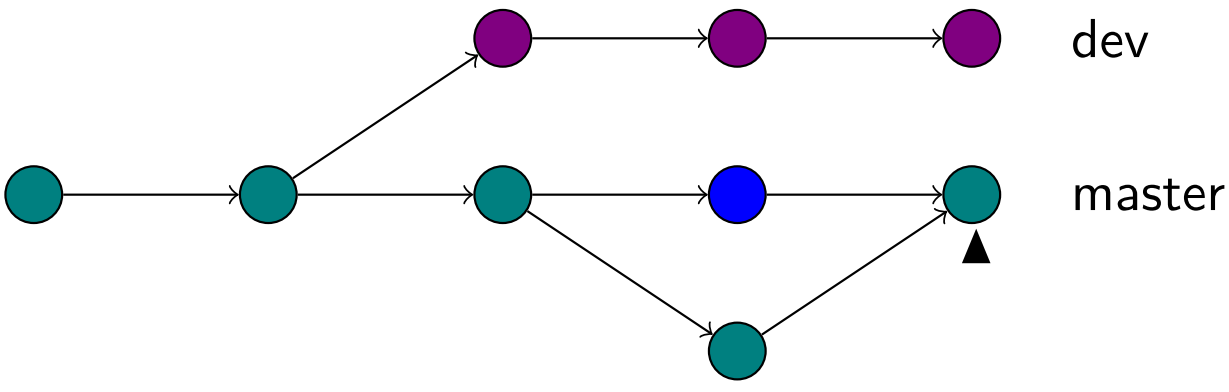


Visualisierung: fetch und merge

`origin`:



Lokal nach `git merge origin/master`:



pull

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

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

Branchmanagement

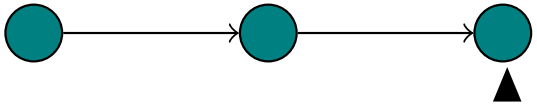
- Manchmal möchte man komplexe Dinge ausprobieren, die nicht mit einer Änderung getan sind
- Dafür kann man einen Branch anlegen
- `:~$ git branch <branch>` legt Branch namens **<branch>** an
- `:~$ git checkout <branch>` wechselt in Branch **<branch>**
- `:~$ git branch -d <branch>` löscht Branch **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

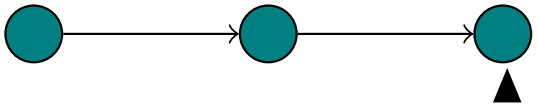
Lokal ursprünglich:



master

Visualisierung: Branches

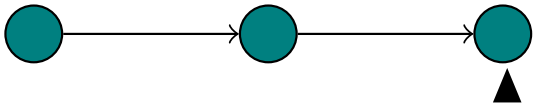
Lokal nach `git branch dev`:



master, dev

Visualisierung: Branches

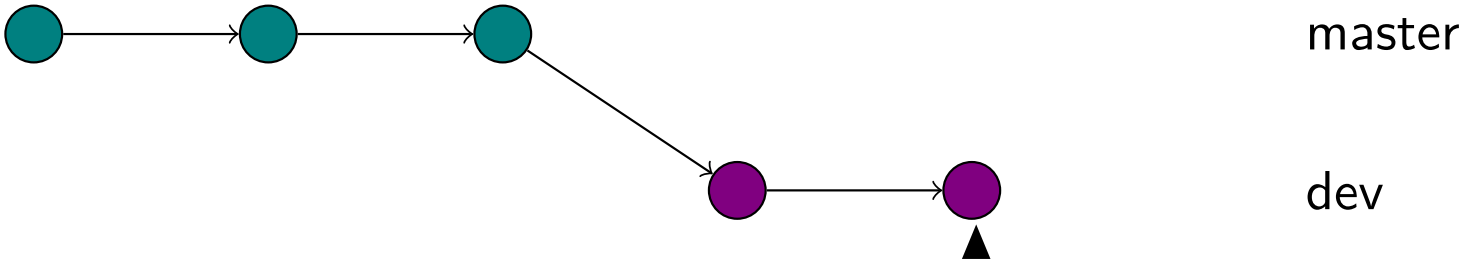
Lokal nach `git checkout dev`:



master, dev

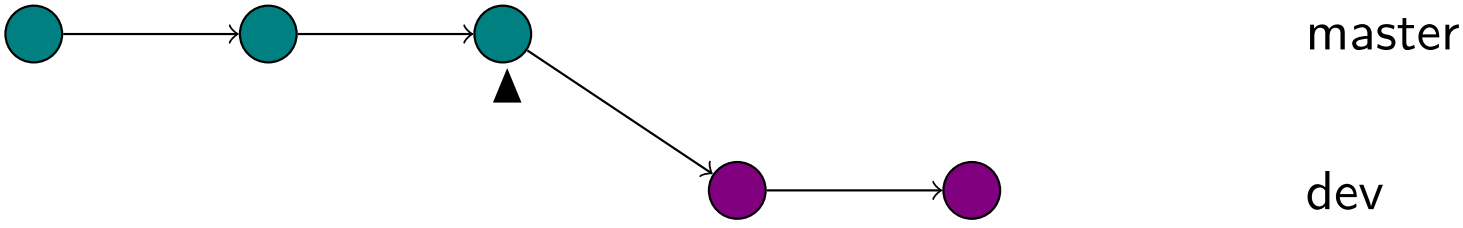
Visualisierung: Branches

Lokal nach Änderungen und mehreren `git commit`:



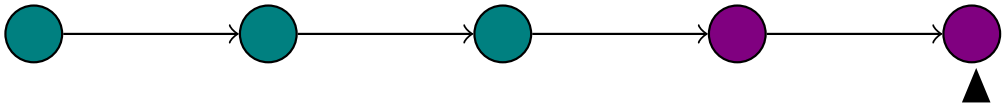
Visualisierung: Branches

Lokal nach `git checkout master`:



Visualisierung: Branches

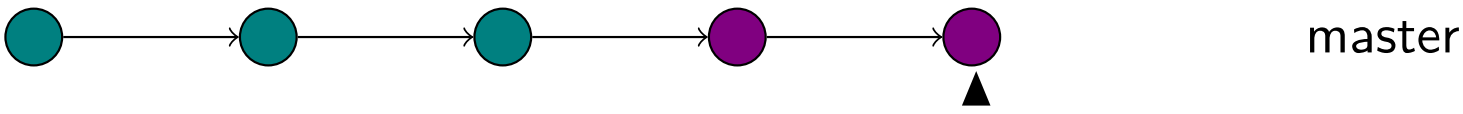
Lokal nach `git merge dev`:



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

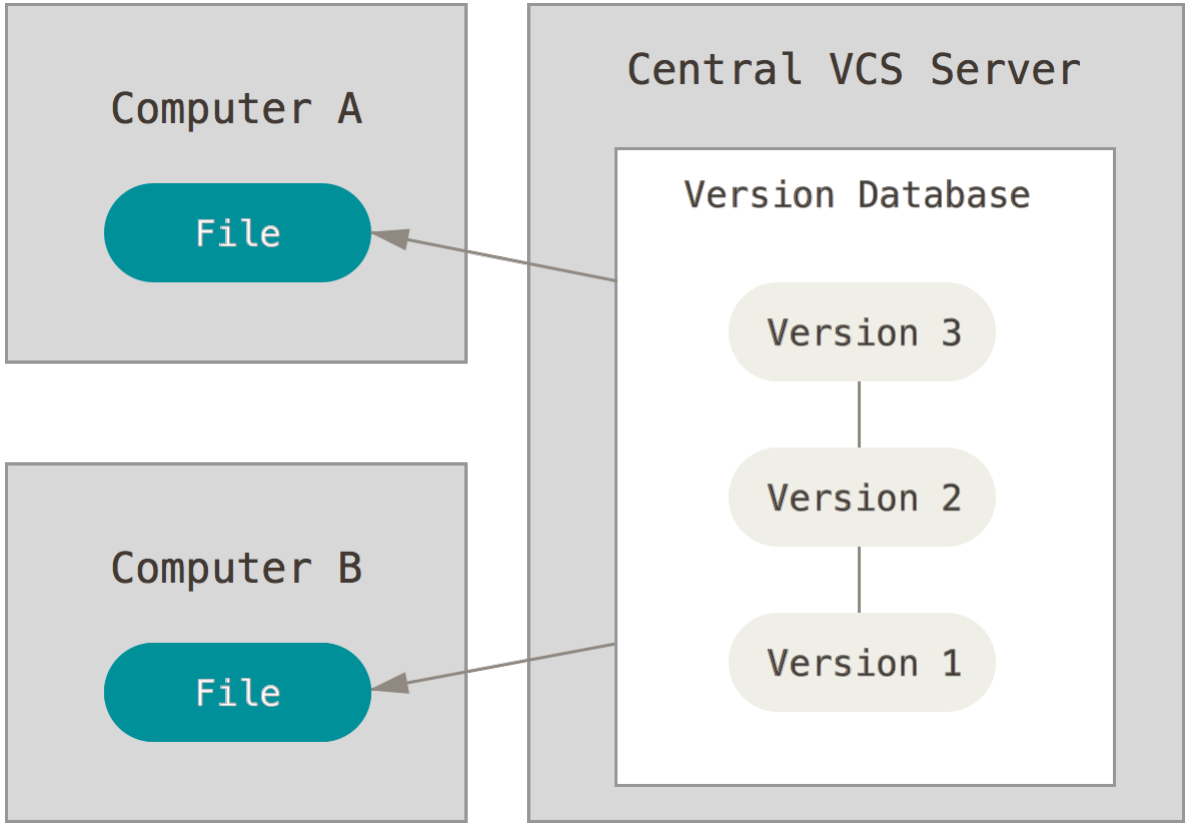
Unerwünschte Dateien

- 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 $\rightarrow$ 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

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

Mittwoch: Parser und Tagger

6 Parser und Tagger
- TreeTagger
- MaltParser
- Stanford Parser

Parser und Tagger

TreeTagger

- Statistischer Part-of-Speech-Tagger
- Frei verfügbar
- `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`
- Viele Erweiterungen
- Auf unseren Servern unter `/resources/processors/tagger/tree-tagger`
- Trainierte Modelle für Englisch, Deutsch, Französisch, Niederländisch, Griechisch, Italienisch, ...

Verzeichnisinhalt

- `bin` – Die eigentlichen Programme
- `cmd` – Shell-Skripte mit eingebautem Preprocessing
- `doc` – Papers
- `lib` – Gespeicherte Modelle

Englisches Tagset

NNS	Noun, plural
NN	Noun, singular or mass
DT	Determiner
JJ	Adjective
MD	Modal verb
VBP	Present tense verb

Vollständige Liste:
`http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/Penn-Treebank-Tagset.pdf`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss `tree-tagger` direkt aufgerufen werden: `tree-tagger`
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus

  `-lemma` Gibt das Lemma aus

  `-prob` Gibt die Wahrscheinlichkeit aus

  `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus

  `-lemma` Gibt das Lemma aus

  `-prob` Gibt die Wahrscheinlichkeit aus

  `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:
  - `-token` Gibt das Token aus
  - `-lemma` Gibt das Lemma aus
  - `-prob` Gibt die Wahrscheinlichkeit aus
  - `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , `tree-tagger-german` , ...
- Um Feineinstellungen vorzunehmen, muss `tree-tagger` direkt aufgerufen werden: `tree-tagger`
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus
  `-lemma` Gibt das Lemma aus
  `-prob` Gibt die Wahrscheinlichkeit aus
  `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus
  `-lemma` Gibt das Lemma aus
  `-prob` Gibt die Wahrscheinlichkeit aus
  `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus
  `-lemma` Gibt das Lemma aus
  `-prob` Gibt die Wahrscheinlichkeit aus
  `-lex f` Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

`-token`	Gibt das Token aus
`-lemma`	Gibt das Lemma aus
`-prob`	Gibt die Wahrscheinlichkeit aus
`-lex f`	Liest ein zusätzliches Lexikon aus Datei `f`

Benutzung

- TreeTagger stellt Voreinstellungen (in Skripten) für die verschiedenen Sprachen bereit:
  `tree-tagger-english` , **`tree-tagger-german`** , ...
- Um Feineinstellungen vorzunehmen, muss **`tree-tagger`** direkt aufgerufen werden: **`tree-tagger`**
- Einstellungen:
  `tree-tagger [OPTIONS] <parameter file> [<input file> [<output file>]]`
- Options:

  `-token` Gibt das Token aus
  `-lemma` Gibt das Lemma aus
  `-prob` Gibt die Wahrscheinlichkeit aus
  `-lex f` Liest ein zusätzliches Lexikon aus Datei **`f`**

Ein- und Ausgabe

`tree-tagger-(english|german|…)`

Der Text muss praktisch nicht vorverarbeitet sein.

`tree-tagger`

Ein Token pro Zeile.

`tree-tagger-german`: Ausgabe

```
      reading parameters ...
      tagging ...
      finished.
Die     ART     d
Sonne   NN      Sonne
scheint VVFIN   scheinen
.       $.      .
```

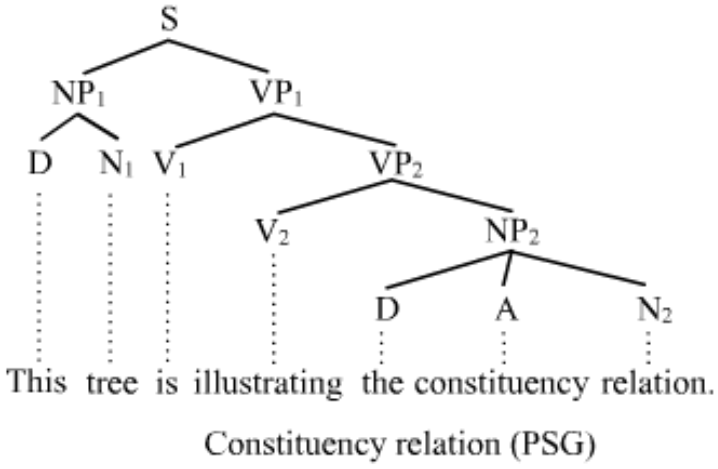
Übung 8

Parser und Tagger

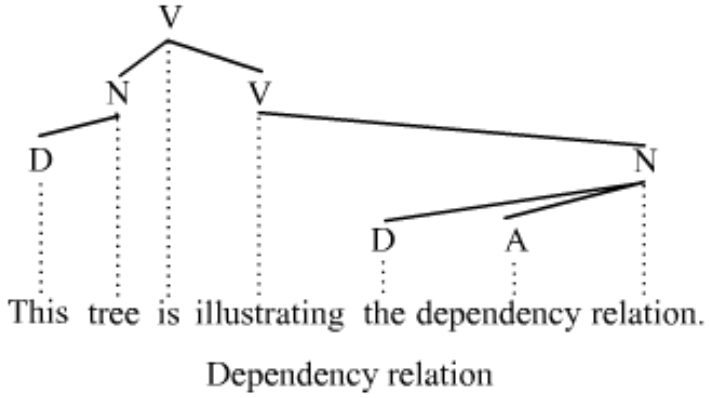
MaltParser

- Deterministischer Dependenz-Parser, implementiert in Java (seit Version 1.0)
- Parsing-Modell wird aus einer Treebank induziert
- Neue Daten werden mit dem induzierten Modell geparst
- Webseite: `http://maltparser.org/download.html`
- Auf unseren Servern unter `/resources/processors/parser/maltparser-1.8/`
- Modelle für Deutsch, Englisch, Französisch und Schwedisch
- Eingabe und Ausgabe im *CoNLL Format*

Dependenz- vs. Konstituenten-Parser



Constituency relation (PSG)

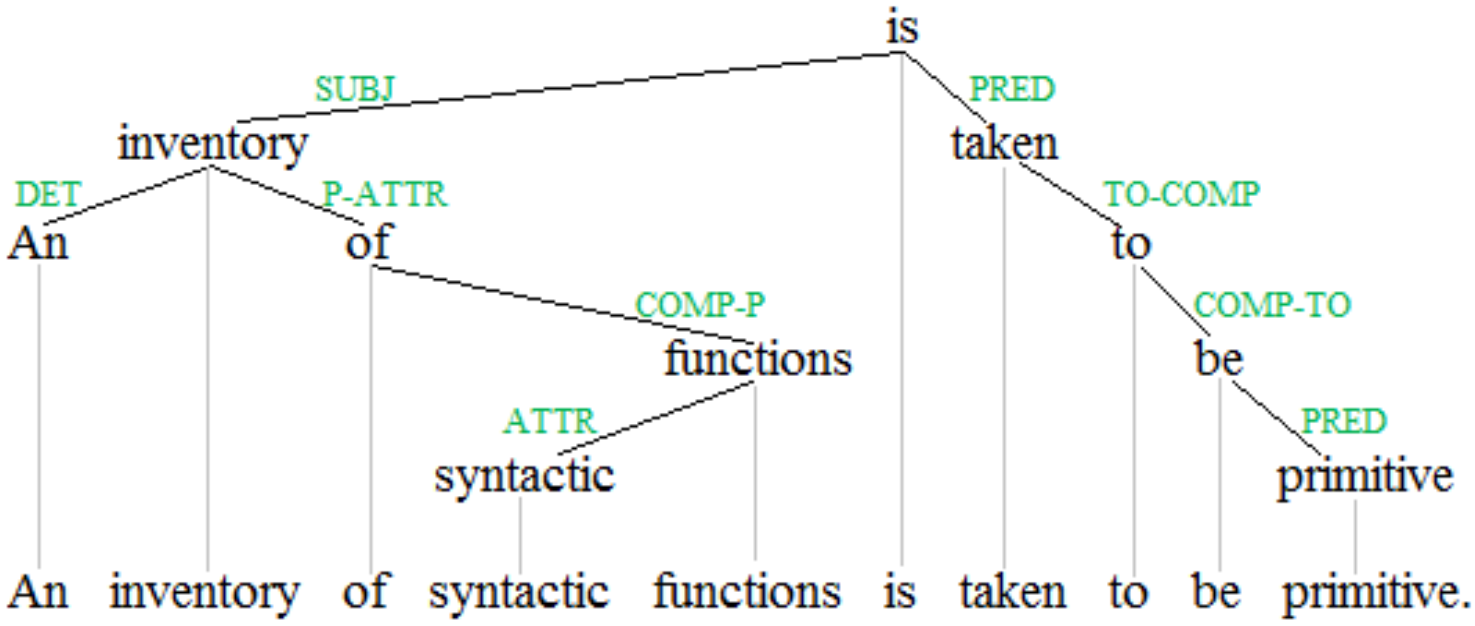


Dependency relation

Quelle:

`https://en.wikipedia.org/wiki/Phrase_structure_grammar`

Dependenzbäume mit Kantenbeschriftungen



Quelle: ByTjo3ya-Ownwork,CCBY-SA3.0,https://commons.wikimedia.org/w/index.php?curid=21976793

CoNLL Format

- Standardisiertes Format für die Eingaben/Ausgaben bei CoNLL Shared Task 2007
- Text-Dateien mit der Erweiterung `.conll`
- Eine Zeile besteht aus 10 durch `TAB` getrennten Feldern
- Ein Satz besteht aus einem oder mehreren Tokens, je eines auf einer Zeile
- Eine Datei kann mehrere Sätze enthalten (getrennt durch eine Leerzeile)
- Leerzeichen in den Feldern sind nicht erlaubt!

`http://ilk.uvt.nl/conll/#dataformat`

CoNLL Format - Felder

1. `ID` - Der Zähler für Tokens (beginnt mit 1 in jedem Satz)
2. `FORM` - Wortform
3. `LEMMA` - Lemma des Wortes
4. `CPOSTAG` - POS-Tag (sprachabhängig)
5. `POSTAG` - Detailliertes POS-Tag (sprachabhängig)
6. `FEATS` - Syntaktische und/oder morphologische Merkmale (sprachabhängig)
7. `HEAD` - Head-Token für das aktuelle Token (ID oder 0)
8. `DEPREL` - Dependenzrelation mit Head-Token (von den Treebank-Annotationen abhängig)
9. `PHEAD` - Projektive Head-Token für das aktuelle Token (ID oder 0)
10. `PDEPREL` - Dependenzrelation mit Head-Token (von den Treebank-Annotationen abhängig)

CoNLL Format - Bemerkungen

- Die folgende Felder *müssen* immer ausgefüllt werden:
  - Bei den Trainingsdaten:
    `ID`, `FORM`, `CPOSTAG`, `POSTAG`, `HEAD`, `DEPREL`
  - Bei den Testdaten:
    `ID`, `FORM`, `CPOSTAG`, `POSTAG`
- Andere Felder können mit dem Platzhalter „_" besetzt werden
- Falls es keine detaillierten POS-Tags für eine bestimmte Sprache gibt, sind die Felder `CPOSTAG` und `POSTAG` gleich
- Mehrere Merkmale werden durch das Symbol „|" getrennt
- Dependenzrelation für den Startknoten ist `ROOT` (mit dem `HEAD`-Wert 0)

CoNLL Format - Beispiel

Satz aus einem Trainingsset

1	Trouble	_	NN	NN	_	2	SBJ	_	_
2	is	_	VBZ	VBZ	_	0	ROOT	_	_
3	,	_	,	,	_	2	P	_	_
4	she	_	PRP	PRP	_	5	SBJ	_	_
5	has	_	VBZ	VBZ	_	2	PRD	_	_
6	lost	_	VBN	VBN	_	5	VC	_	_
7	it	_	PRP	PRP	_	6	OBJ	_	_
8	just	_	RB	RB	_	6	MNR	_	_
9	as	_	RB	RB	_	8	AMOD	_	_
10	quickly	_	RB	RB	_	8	AMOD	_	_
11	.	_	.	.	_	2	P	_	_

CoNLL Format - Beispiel

Satz aus einem Testset

```
1   Pierre        _   NNP   NNP   _
2   Vinken        _   NNP   NNP   _
3   ,             _   ,     ,     _
4   61            _   CD    CD    _
5   years         _   NNS   NNS   _
6   old           _   JJ    JJ    _
7   ,             _   ,     ,     _
8   will          _   MD    MD    _
9   join          _   VB    VB    _
10  the           _   DT    DT    _
11  board         _   NN    NN    _
12  as            _   IN    IN    _
13  a             _   DT    DT    _
14  nonexecutive  _   JJ    JJ    _
15  director      _   NN    NN    _
16  Nov.          _   NNP   NNP   _
17  29            _   CD    CD    _
18  .             _   .     .     _
```

CoNLL Format - Beispiel

Andere Sprache

1	Grundavdraget	_	N	NN	DD|SS	2	SS	_	_
2	blir	_	V	BV	PS	0	ROOT	_	_
3	2500	_	R	RO	_	4	DT	_	_
4	kr	_	N	NN	_	2	SP	_	_
5	(	_	I	IR	_	0	ROOT	_	_
6	=	_	I	IG	_	15	DT	_	_
7	4500	_	R	RO	_	8	DT	_	_
8	minskat	_	P	TP	PA	0	ROOT	_	_
9	med	_	PR	PR	_	14	AA	_	_
10	1/5	_	N	NN	_	9	PA	_	_
11	av	_	PR	PR	_	10	ET	_	_
12	10000	_	R	RO	_	13	DT	_	_
13	kr	_	N	NN	_	11	PA	_	_
14	)	_	I	IR	_	0	ROOT	_	_
15	.	_	P	IP	_	2	IP	_	_

Erster Start des MaltParsers

- Starten des MaltParsers ohne Argumente:

```
:~$ java -jar /path/to/maltparser-1.8/maltparser-1.8.jar
```

  - Gibt die Hauptoptionen für den MaltParser aus

- Starten der Hilfe für den MaltParser:

```
:~$ java -jar /path/to/maltparser-1.8/maltparser-1.8.jar -h
```

  - Gibt alle Optionen für den MaltParser aus

Den MaltParser trainieren

```
:~$ java -jar /path/to/maltparser-1.8/maltparser-1.8.jar -c model \
-i training-file.conll -m learn
```

- **`model`** : Name des Modells
  Achtung: nur den Namen ohne Pfad angegeben, die Erweiterung (**`.mco`**) wird *automatisch* angefügt!
- **`training-file.conll`** : Datei mit den Trainingsdaten im CoNLL-Format
- **`-m learn`** : der Parser wird im Lern-Modus gestartet

Das erstellte Modell wird (hier) in einer Datei namens **`model.mco`** im aktuellen Verzeichnis gespeichert.

Vortrainierte Modelle für den MaltParser

- Für diejenigen, die nicht mit unterschiedlichen Algorithmen und Modellen experimentieren möchten, sondern nur einen Dependenzparser brauchen
- Aktuell sind drei fertige Modelle vorhanden: **`engmalt`**, **`fremalt`** und **`swemalt`** (alle jeweils **`*.linear`** und **`*.poly`**)
- Download: `http://maltparser.org/mco/mco.html`
- Basieren auf Penn Treebank, French Treebank und Swedisch Treebank

Parsen von Daten im MaltParser

```
:~$ java -jar /path/to/maltparser-1.8/maltparser-1.8.jar -c model\
-i test-file.conll -o result-file.conll -m parse
```

- **`model`** : Name des Modells
  Achtung: nur den Namen ohne die Erweiterung (**`.mco`**) angeben
- **`test-file.conll`** : Datei mit den Testdaten
- **`result-file.conll`** : Datei mit den Ergebnissen des Parsers
- **`-m parse`** : der Parser wird im Parsing-Modus gestartet

MaltParser – Weitere Informationen

- Userguide: `http://maltparser.org/userguide.html`
- (Technische) Informationen zur Optimierung:
  `http://www.maltparser.org/guides/opt/quick-opt.pdf`

Übung 9

Parser und Tagger

Stanford Parser

- Java-Implementierung eines probabilistischen NLP-Parser
- Kann sowohl Dependenzstrukturen als auch Phrasenstrukturen ausgeben
- GNU General Public License
- Download: `http://nlp.stanford.edu/software/lex-parser.shtml`
- Auf unseren Servern unter `/resources/processors/parser/stanfordparser-3.6.0`
- Modelle für Englisch, Deutsch, Französisch, Chinesisch, Arabisch, Italienisch, Bulgarisch, Portugiesisch, ...
- Online-Version: `http://nlp.stanford.edu:8080/parser/`

Online Demo-Version

Siehe Browser.

Dependenzen im Stanford Parser

- Mehrere Formalismen verfügbar
- Alle Dependenzen sind binäre Relationen
- Tokens werden zusammen mit dem Index gezeigt
  Beispiel:
  `nsubj(makes-8, Bell-1)`
- Standard: *Universal Dependencies*
  - Insgesamt: 40 grammatische Funktionen
  - sprachunabhängig

Dependenztypen

- Universal Dependencies - alle Dependenzen werden individuell dargestellt
  Beispiel:
  `prep(based-7, in-8)`
  `pobj(in-8, LA-9)`
- Enhanced Universal Dependencies - u.a.: manche Dependenzen (z.B. mit Präpositionen, Konjunktionen) werden zusammengefasst.
  Beispiel:
  `prep:in(based-7, LA-9)`

Eingabe-Datei

- Einfache Textdatei
- Ein oder mehrere Sätze pro Zeile (keine Leerzeilen zwischen den Zeilen)
- Mehrere Sätze in einer Zeile werden mit einem Sentence Splitter getrennt

Beispiel:

```
The strongest rain ever recorded in India shut down
the financial hub of Mumbai, snapped communication
lines, closed airports and forced thousands of people
to sleep in their offices or walk home during the
night, officials said today.
```

Generierung der Ausgabe

```
:~$ java -mx200m edu.stanford.nlp.parser.lexparser.\
LexicalizedParser -outputFormat "wordsAndTags,penn,typedDependencies
"englishPCFG.ser.gz text
```

- **`englishPCFG.ser.gz`** : Vortrainiertes Modell für Englisch
- **`-outputFormat`** : Format für die Ausgabe (Mehrere Optionen durch Kommata getrennt)
- **`text`** : Eingabe-Datei mit den Sätzen
- **`-sentences newline`** : optional. Unterdrückt das Starten des Sentence Splitters

Das Ergebnis wird auf STDOUT ausgegeben.

Das Ausgabeformat

Die Optionen für `outputFormat` im Beispiel:

- `wordsAndTags`: Text mit POS-Tags
- `penn`: Syntaktische Baumstrukturen
- `typedDependencies`: Grammatische Relationen in Dependenzformat (Enhanced Universal Dependencies)

Ausgabe-Datei – Teil 1 und 2

```
The/DT strongest/JJS rain/NN ever/RB recorded/VBN in/IN
India/NNP shut/VBD down/RP the/DT financial/JJ hub/NN
of/IN Mumbai/NNP ,/, snapped/VBD communication/NN lines/NNS
,/, closed/VBD airports/NNS and/CC forced/VBD thousands/NNS
of/IN people/NNS to/TO sleep/VB in/IN their/PRP$ offices/NNS
or/CC walk/VB home/NN during/IN the/DT night/NN ,/,
officials/NNS said/VBD today/NN ./.
```

```
(ROOT
  (S
    (S
      (NP
        (NP (DT The) (JJS strongest) (NN rain))
        (VP
          (ADVP (RB ever))
          (VBN recorded)
          (PP (IN in)
            (NP (NNP India)))))
      (VP
        (VP (VBD shut)
          (PRT (RP down))
          (NP
            (NP (DT the) (JJ financial) (NN hub))
            (PP (IN of)
              (NP (NNP Mumbai)))))
        (, ,)
```

Ausgabe-Datei – Teil 2

```
      (VP (VBD snapped)
        (NP (NN communication) (NNS lines)))
      (, ,)
      (VP (VBD closed)
        (NP (NNS airports)))
      (CC and)
      (VP (VBD forced)
        (NP
          (NP (NNS thousands))
          (PP (IN of)
            (NP (NNS people))))
        (S
          (VP (TO to)
            (VP
              (VP (VB sleep)
                (PP (IN in)
                  (NP (PRP$ their) (NNS offices))))
              (CC or)
              (VP (VB walk)
                (NP (NN home))
                (PP (IN during)
                  (NP (DT the) (NN night)))))))))
  (, ,)
  (NP (NNS officials))
  (VP (VBD said)
    (NP (NN today)))
  (. .)))
```

Ausgabe-Datei – Teil 3 I

```
det(rain-3, The-1)
amod(rain-3, strongest-2)
nsubj(shut-8, rain-3)
nsubj(snapped-16, rain-3)
nsubj(closed-20, rain-3)
nsubj(forced-23, rain-3)
advmod(recorded-5, ever-4)
acl(rain-3, recorded-5)
case(India-7, in-6)
nmod:in(recorded-5, India-7)
ccomp(said-40, shut-8)
compound:prt(shut-8, down-9)
det(hub-12, the-10)
amod(hub-12, financial-11)
dobj(shut-8, hub-12)
case(Mumbai-14, of-13)
nmod:of(hub-12, Mumbai-14)
conj:and(shut-8, snapped-16)
ccomp(said-40, snapped-16)
compound(lines-18, communication-17)
```

Ausgabe-Datei – Teil 3 II

```
dobj(snapped-16, lines-18)
conj:and(shut-8, closed-20)
ccomp(said-40, closed-20)
dobj(closed-20, airports-21)
cc(shut-8, and-22)
conj:and(shut-8, forced-23)
ccomp(said-40, forced-23)
dobj(forced-23, thousands-24)
nsubj(sleep-28, thousands-24)
nsubj(walk-33, thousands-24)
case(people-26, of-25)
nmod:of(thousands-24, people-26)
mark(sleep-28, to-27)
xcomp(forced-23, sleep-28)
case(offices-31, in-29)
nmod:poss(offices-31, their-30)
nmod:in(sleep-28, offices-31)
cc(sleep-28, or-32)
xcomp(forced-23, walk-33)
conj:or(sleep-28, walk-33)
dobj(walk-33, home-34)
case(night-37, during-35)
det(night-37, the-36)
```

Ausgabe-Datei – Teil 3 III

```
nmod:during(walk-33, night-37)
nsubj(said-40, officials-39)
root(ROOT-0, said-40)
nmod:tmod(said-40, today-41)
```

Notationen im Stanford Parser

- Menge der POS-Tags und -Kategorien
  - Englisch – aus Penn Treebank
  - Chinesisch – aus Penn Chinese Treebank
  - Deutsch – aus NEGRA Korpus
- Menge der grammatischen Funktionen
  - Universal Dependecies
    `http://universaldependencies.org/`
  - ältere, sprachspezifische Formalismen auch noch verfügbar

Wichtige Dependenzen I

- **`nsubj`**: nominal subject
  - Subjekt in einer Phrase
  - Hauptwort ist nicht unbedingt ein Verb (z.B. *The flower is blue* - **`nsubj(blue,flower)`**)
- **`nsubjpass`**: passive nominal subject
  - Subjekt in einer Phrase mit dem Verb im Passiv

Wichtige Dependenzen II

- `dobj`: direct object
  - Direktes Objekt in einer Phrase
  - Immer im Akkusativ
- `iobj`: indirect object
  - Indirektes Objekt in einer Phrase
  - Immer im Dativ

Wichtige Dependenzen III

- `advmod`: adverbial modifier
  - Adverbialer Modifikator
- `amod`: adjectival modifier
  - Adjektivaler Modifikator
- `nmod`: nominal modifier
  - Nominaler Modifikator

Parser-Ausgabe exportieren

- Man kann Konstituentenparsebäume in das CoNLL-Format exportieren
- Option: **`-conllx`**
- Mit **`-keepPunct`** wird die Interpunktion behalten
- Beispiele:

```
:~$ java edu.stanford.nlp.trees.UniversalEnglishGrammatical\
Structure -treeFile file.tree -collapsedTree -conllx -keepPunct
```

Weitere Stanford NLP Tools

Stanford bietet nicht nur den Parser

- POS Tagger
  `http://nlp.stanford.edu/software/tagger.shtml`
- Named Entity Recognizer
  `http://nlp.stanford.edu/software/CRF-NER.shtml`
- Word Segmenter
  `http://nlp.stanford.edu/software/segmenter.shtml`
- ...
- zusamengefasst in CoreNLP:
  `http://stanfordnlp.github.io/CoreNLP/`

Übung 10

Mittagspause

Mittwoch: Python für NLP

7 Python für NLP
- NLTK
- spaCy

Python für NLP

Was ist NLTK?

- Zusammenstellung von Python-Modulen für NLP-Lehre, -Forschung und -Entwicklung
- Code für Vielzahl von Aufgaben aus der Sprachverarbeitung
- 40 bekannte Korpora
- ausführliche Dokumentation inkl. Online-Buch

Ein einfaches Beispiel

- Einbindung in Python-Programme:

```
>>> text = 'NLTK is my favourite natural language toolkit.'
>>> import nltk
>>> nltk.WhitespaceTokenizer().tokenize(text)
['NLTK', 'is', 'my', 'favourite', 'natural', 'language',
 'toolkit.']
```

Was NLTK bietet

- Infrastruktur als Grundlage für NLP-Programme in Python:
  1. Grundlegende Klassen zur Repräsentation NLP-relevanter Daten
  2. Standard-Schnittstellen für übliche Aufgaben, z.B. Tokenisierung, Tagging, Parsing
  3. Demos, z.B. Parser, Chunker, Chatbots
  4. Ausführliche Dokumentation, Tutorien, Referenzen

Dokumentation

- Online-Buch: `http://www.nltk.org/book/`
- API-Dokumentation: `http://www.nltk.org/api/`

Hilfe

- Online-Hilfe-Funktion:

```
>>> help(nltk.stem.WordnetStemmer)
class WordnetStemmer(nltk.stem.api.StemmerI)
 |  A stemmer that uses Wordnet`s built-in morphy function.
 |
 |  Method resolution order:
 |      WordnetStemmer
 |      nltk.stem.api.StemmerI
...
```

- Doc-Strings:

```
>>> print nltk.tag.BrillTagger.__doc__

    Brill's transformational rule-based tagger.  Brill
    taggers use an X{initial tagger} (such as
    L{tag.DefaultTagger}) to assign an intial tag
    sequence to a text; and then apply an ordered list
    of ...
```

Module

- NLTK ist in *Module* organisiert, die unterschiedliche Funktionalität bereitstellen.
- Wir schauen uns einige Module beispielhaft genauer an.

Modul `grammar` I

- Klassen zur Repräsentation kontextfreier Grammatiken (CFGs), Regeln, Nichtterminalzeichen, probabilistischer CFGs
- Funktionen zum Prüfen der Abdeckung, Induzieren probabilistischer CFGs, Parsen von Grammatiken, ...

Modul grammar II

```
>>> from nltk import grammar
>>> gr = grammar.CFG.fromstring("""
... S -> NP VP
... PP -> P NP
... NP -> Det N | NP PP
... VP -> V NP | VP PP
... Det -> 'a' | 'the'
... N -> 'dog' | 'cat'
... V -> 'chased' | 'sat'
... P -> 'on' | 'in'
... """)
>>> gr
<Grammar with 14 productions>
>>> gr.check_coverage("a cat chased a mouse".split())
ValueError: Grammar does not cover some of the input
words: u"'mouse'".
```

http://www.nltk.org/howto/grammar.html

Modul `corpus`

- Korpus-Reader für 40 NLTK-interne & -externe Korpora
- Beispiele für Funktionen einzelner Reader: `word()`, `sents()`, `paras()`, `tagged_words()`, `chunked_sents()`, `parsed_paras()`, `raw()`

```
>>> from nltk.corpus import brown
>>> print brown.words()
['The', 'Fulton', 'County', 'Grand', 'Jury',
 'said', ...]
```

Modul tag

- Klassen für verschiedene Taggermodelle: AffixTagger, BrillTagger, NGramTagger, HMM-Tagger

```
>>> from nltk import corpus, tag
>>> tagger = tag.UnigramTagger(corpus.brown.tagged_sents(),
...                            backoff=tag.DefaultTagger('NN'))
>>> tagger.tag(['The', 'dog', 'chewed', 'on', 'a', 'bone', '.'])
[('The', 'AT'), ('dog', 'NN'), ('chewed', 'VBD'), ('on', 'IN'),
('a', 'AT'), ('bone', 'NN'), ('.', '.')]
```

Modul `probability`

- Klassen zur Berechnung von Statistiken über Ergebnisse von Experimenten, unter anderem
  - FreqDist: Häufigkeitsverteilung
  - ConditionalFreqDist: Bedingte Häufigkeitsverteilungen, z.B.: Wie oft wird ein Wort mit welchem Tag versehen?

```
>>> from nltk import probability, tokenize
>>> probability.FreqDist(
...     tokenize.word_tokenize("this is a sentence"))
FreqDist({'this': 1, 'a': 1, 'is': 1, 'sentence': 1})
```

Beispiel: Satzgenerierung I

(Beispiele entnommen aus Madnani (2007): `http://www.umiacs.umd.edu/~nmadnani/pdf/crossroads.pdf`)

```
>>> from nltk.corpus import gutenberg
>>> from nltk.probability import ConditionalFreqDist
>>> from random import choice

# Create distribution object
>>> cfd = ConditionalFreqDist()

# For each token, count current word given previous word
>>> prev_word = None
>>> for word in gutenberg.words('austen-persuasion.txt'):
...     cfd[prev_word][word] += 1
...     prev_word = word
```

Beispiel: Satzgenerierung II

```
# Start predicting at the given word, say 'therefore'
>>> word = 'therefore'
>>> i = 1

# Find all words that can possibly follow the current word
# and choose one at random
>>> while i < 20:
...     print word
...     lwords = []
...     for next_word, count in cfd[word].items():
...         lwords.extend([next_word]*count)
...     follower = choice(lwords)
...     word = follower
...     i += 1

therefore it known of women ought . Leave me so well
placed in five altogether well placed themselves delighted
```

Weitere Module

Weitere Module sind unter anderem:

- classify
- metrics
- parse
- sem
- sentiment

Übung 11

Python für NLP

Was ist spaCy?

- Python-Module für NLP-Entwicklung
- Fokus auf Geschwindigkeit und State-of-the-Art-Performanz
- Konzentration auf wenige essentielle Aufgaben
- für jede Aufgabe nur ein Algorithmus implementiert
- unterstützte Sprachen: Englisch, Deutsch, Französisch, Spanisch

Ein Beispiel

```
>>> import spacy
>>> nlp = spacy.load("en")
>>> doc = nlp(u"I love computational linguistics!")
>>> for token in doc:
...     print token, token.tag_, token.prob

I PRP -4.064180850982666
love VBP -7.974212169647217
computational JJ -19.579313278198242
linguistics NNS -19.579313278198242
! . -5.515825748443603 5
```

NLTK vs spaCy

NLTK:

- gut geeignet für Lehre
- enthält viele Algorithmen und Schnittstellen
- gut dokumentiert

spaCy:

- Idee: “getting things done”
- schneller und bessere Ergebnisse als NLTK
- kann sehr gut im Pre-Processing verwendet werden (auch für Deutsch)!

Dokumentation

- API-Dokumentation, Beispiele und Tutorials:
  `https://spacy.io/docs/`

Daten in spaCy

```
>>> doc = nlp(u"I love computational linguistics!")
```

- Annotationen gespeichert in Objekt der `Doc`-Klasse
- einzelne Wörter: Objekte der `Token`-Klasse
- Zugriff auf Wörter/Sequenzen: `doc[i]`, `doc[(i, j)]`
- annotiert: Tokenisierung, Satz-Splitting, POS-Tags, Dependenzbäume, Named Entity Recognition, Sentiment
- weitere Informationen: Wahrscheinlichkeit nach Language Model, Wort-Vektoren

Informationen auf Wortebene

```
>>> doc = nlp(u"I love computational linguistics!")
>>> print doc[1].tag_
```

Alle Attribute: `https://spacy.io/docs/api/token`

Informationen auf Dokumentenebe

```
>>> doc = nlp(u"I love computational linguistics!")
>>> print doc.vector
```

Alle Attribute: `https://spacy.io/docs/api/doc`

Aufbau I

- man muss zunächst die Pipeline laden:

```
>>> import spacy
>>> nlp = spacy.load("en")
```

- die Pipeline erwartet Text(e) als Unicode-Strings:

```
>>> doc = nlp(u"""I love computational linguistics!
Green frogs are green""")
>>> docs = nlp.pipe(list_of_texts)
```

Aufbau II

- dann kann man über Dokumente, Sätze und Token iterieren:

```
>>> for token in doc:
...     print(token.lemma_)
```

```
>>> other_doc = nlp(u"I hate geometry.")
>>> for sentence in doc.sents:
...     print sentence, sentence.similarity(other_doc)
I love computational linguistics! 0.839319268367
Green frogs are green. 0.646078181155
```

Vorhandene Annotationen

- schon vorhandene Annotation zu nutzen ist auch möglich:

```
>>> import spacy
>>>
>>> nlp = spacy.load("en")
>>>
>>> tokens = [u"Just", u"some", u"example"]
>>> doc = spacy.tokens.Doc(nlp.vocab, words=tokens)
>>> nlp.tagger(doc)
>>> for tok in doc:
...     print tok.pos_, tok.tag_
```

https://spacy.io/docs/usage/processing-text

Übung 12

Donnerstag: Python für Machine Learning

8 Python für Machine Learning
- Profiling
- Numpy und SciPy
- Die Numpy Package
- Die SciPy-Library
- scikit-learn

Python für Machine Learning

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

---

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Modul `timeit`

http://docs.python.org/library/timeit.html

- Modul, um kleine Pythoncodestücke zu timen.
- Handhabung im Interpreter:

```
>>> import timeit
>>> setup = "l = range(10000)" # parameters
>>> stmt = "len(l)" # statement to evaluate
>>> timeit.timeit(stmt=stmt, setup=setup, number=10000)
0.0019829273223876953
```

- `stmt`: Ausdruck, dessen benötigte Zeit gemessen werden soll
- `setup`: Anweisungen, die (einmal) vor der Ausführung von `stmt` durchgeführt werden sollen
- `number`: Anzahl der Ausführungen von `stmt` (Default 1000000)

Rückgabe: Benötigte Zeit in Sekunden

Modul cProfile

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

- `timeit` ist sehr umständlich, wenn man Funktionen oder Programme evaluieren will.
- `cProfile` ist ein in der Standardbibliothek eingebautes Werkzeug, das misst, wie oft und wie lange Teile des Programms ausgeführt wurden.
- Warnung: Profiling verlängert die Laufzeit
- Profiler auf ein Skript anwenden:

```
$ python -m cProfile myProgram.py
```

Beispiel: Tiny Search Engine

Idee

- Implementierung einer Mini-Suchmaschine
- Die Suchanfrage (*query*) und die Dokumente (*documents*) sind Worthäufigkeitsvektoren.
- Ähnlichkeit zwischen Suchanfrage und Dokument: Kosinus

Tiny Search Engine: Pseudodaten

Anstatt echte Dokumente einzulesen, werden für Suchanfrage und Dokumente zufällig “Termhäufigkeiten” zwischen 0 und 100 gezogen:

```
    v = []
    for i in range(l):
        freq = random.randint(0, 100)
        v.append(freq)
    return v
```

Tiny Search Engine: Kosinusähnlichkeit

$$\cos(a, b) = \frac{a \cdot b}{\|a\|\|b\|} = \frac{a \cdot b}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}}$$

Einfache Python-Implementierung

```
def cosine(a, b):
    assert(len(a)==len(b))
    ab_sum = 0
    a_sq_sum = 0
    b_sq_sum = 0
    for i in range(len(a)):
        ab_sum += a[i] * b[i]
        a_sq_sum += a[i] * a[i]
        b_sq_sum += b[i] * b[i]
    total = ab_sum / (math.sqrt(a_sq_sum) * math.sqrt(b_sq_sum))
    return total
```

Tiny Search Engine: run

```
    random.seed()
    query = pseudo_data(vocab_size)
    maxsim = 0
    best = []
    for i in range(ndocs):
        doc = pseudo_data(vocab_size)
        cos = cs.cosine(query, doc)
        if cos > maxsim:
            maxsim = cos
            best = doc

    return (maxsim, best)
```

cProfile Aufruf

```
$ python -m cProfile -s cumtime \
tiny_search_engine.py --vocab-size 10000 \
--num-docs 1000
```

`-s`: Sortierung der Ausgabe (hier nach kumulativer Laufzeit)

Ausgegebene Statistiken (in Spalten)

ncalls	Anzahl der Aufrufe
tottime	Gesamtlaufzeit dieser Funktion (ohne Subfunktionen)
percall	tottime / ncalls
cumtime	Gesamtlaufzeit dieser Funktion (mit Subfunktionen)
percall	cumtime / ncalls

cProfile Aufruf

```
$ python -m cProfile -s cumtime \
tiny_search_engine.py --vocab-size 10000 \
--num-docs 1000
```

`-s`: Sortierung der Ausgabe (hier nach kumulativer Laufzeit)

Ausgegebene Statistiken (in Spalten)

ncalls	Anzahl der Aufrufe
tottime	Gesamtlaufzeit dieser Funktion (ohne Subfunktionen)
percall	tottime / ncalls
cumtime	Gesamtlaufzeit dieser Funktion (mit Subfunktionen)
percall	cumtime / ncalls

cProfile Ausgabe

```
         40092169 function calls (40091230 primitive calls) in 19.384 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   19.385   19.385 tiny_search_engine.py:3(<module>)
        1    0.000    0.000   19.282   19.282 tiny_search_engine.py:109(main)
        1    0.016    0.016   19.281   19.281 tiny_search_engine.py:44(run)
     1001    3.987    0.004   17.195    0.017 tiny_search_engine.py:33(pseudo_data)
 10010000    3.300    0.000   12.412    0.000 random.py:238(randint)
 10010000    8.321    0.000    9.111    0.000 random.py:175(randrange)
     1000    2.012    0.002    2.070    0.002 cosinus.py:8(cosine)
 10010000    0.791    0.000    0.791    0.000 {method 'random' of '_random.Random' objects}
 10015034    0.743    0.000    0.743    0.000 {method 'append' of 'list' objects}
     2113    0.111    0.000    0.111    0.000 {range}
```

...

Visualisierung der Profiler-Ausabe

1. Profiler-Ausgabe in Datei schreiben:

```
$ python -m cProfile -o profile.stats
tiny_search_engine.py -v 10000 -d 1000
```

2. gprof2dot: Python-Skript, um Profiler-Ausgabe in dot-Format zu übertragen.[2]

```
$ gprof2dot profile.stats -f pstats > profile.dot
```

3. dot: Tool zum Zeichnen von gerichteten Graphen

```
$ dot profile.dot -Tpng -o profile.png
```

Andere Pakete für Visualisierung: z.B. `snakeviz`, `pycallgraph`

---

[2] siehe `https://github.com/jrfonseca/gprof2dot`. Mehr zum dot-Format: `http://www.graphviz.org/doc/info/lang.html`

Visualisierung der Profiler-Ausabe

1. Profiler-Ausgabe in Datei schreiben:

```
$ python -m cProfile -o profile.stats
tiny_search_engine.py -v 10000 -d 1000
```

2. `gprof2dot`: Python-Skript, um Profiler-Ausgabe in dot-Format zu übertragen.[2]

```
$ gprof2dot profile.stats -f pstats > profile.dot
```

3. `dot`: Tool zum Zeichnen von gerichteten Graphen

```
$ dot profile.dot -Tpng -o profile.png
```

Andere Pakete für Visualisierung: z.B. `snakeviz`, `pycallgraph`

[2] siehe `https://github.com/jrfonseca/gprof2dot`. Mehr zum dot-Format: `http://www.graphviz.org/doc/info/lang.html`

Visualisierung der Profiler-Ausabe

1. Profiler-Ausgabe in Datei schreiben:

```
$ python -m cProfile -o profile.stats
tiny_search_engine.py -v 10000 -d 1000
```

2. `gprof2dot`: Python-Skript, um Profiler-Ausgabe in dot-Format zu übertragen.[2]

```
$ gprof2dot profile.stats -f pstats > profile.dot
```

3. `dot`: Tool zum Zeichnen von gerichteten Graphen

```
$ dot profile.dot -Tpng -o profile.png
```

Andere Pakete für Visualisierung: z.B. `snakeviz`, `pycallgraph`

[2] siehe `https://github.com/jrfonseca/gprof2dot`. Mehr zum dot-Format: `http://www.graphviz.org/doc/info/lang.html`

Visualisierung der Profiler-Ausabe

1. Profiler-Ausgabe in Datei schreiben:

```
$ python -m cProfile -o profile.stats
tiny_search_engine.py -v 10000 -d 1000
```

2. `gprof2dot`: Python-Skript, um Profiler-Ausgabe in dot-Format zu übertragen.[2]

```
$ gprof2dot profile.stats -f pstats > profile.dot
```

3. `dot`: Tool zum Zeichnen von gerichteten Graphen

```
$ dot profile.dot -Tpng -o profile.png
```

Andere Pakete für Visualisierung: z.B. `snakeviz`, `pycallgraph`

---

[2] siehe `https://github.com/jrfonseca/gprof2dot`. Mehr zum dot-Format: `http://www.graphviz.org/doc/info/lang.html`

Callgraph

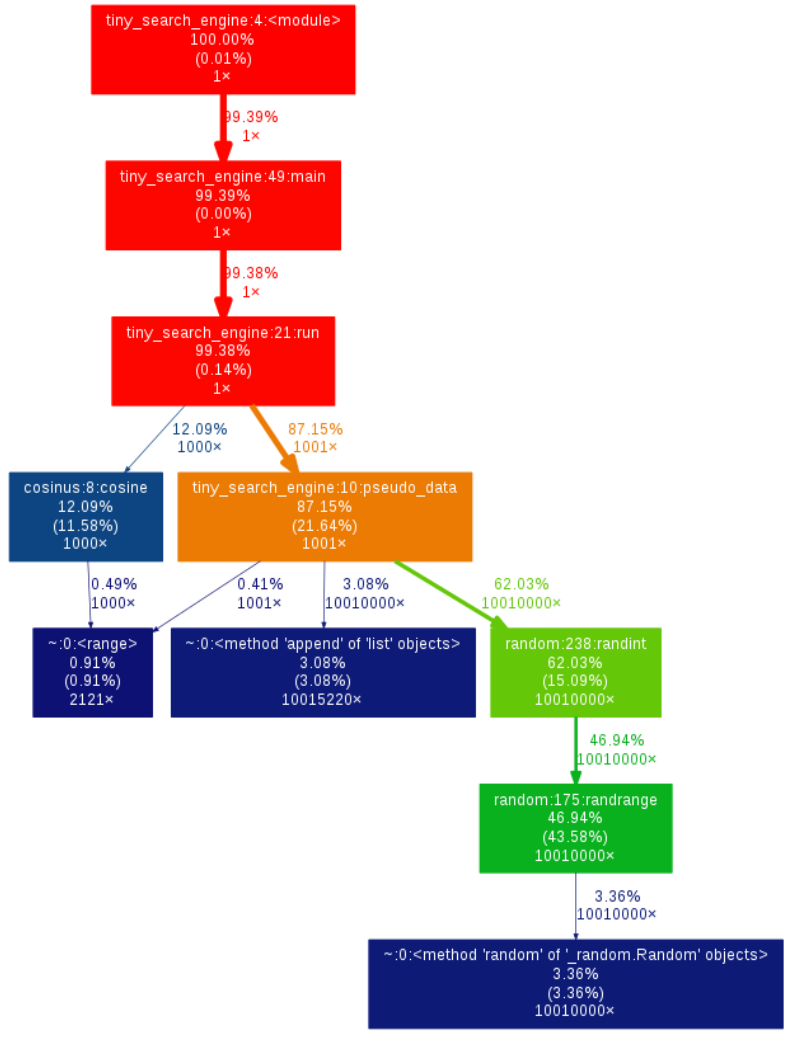


Abbildung: Callgraph für `tiny_search_engine.py`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: **`sudo pip install line_profiler`**

Aufruf

Auf der Kommandozeilen:

```
$ kernprof -l -v tiny_search_engine.py
```

`-l` Aktiviert den Line Profiler.

`-v` Ausgabe wird angezeigt.

Default Ausgabe wird in (Binär-)datei `<Scriptname>.py.lprof` geschrieben.

Line Profiler anwenden

- Um den `line_profiler` auf eine Funktion anzuwenden, müssen wir ihm das erst mitteilen
- Dazu "dekoriert" man die Funktion mit `@profile`. Beispiel:

```
@profile
def pseudo_data(l):
    v = []
    for i in range(l):
        v.append(random.randint(0,100))
    return v
```

Decorators

- Ein Decorator ist ein aufrufbares Python-Objekt, das auf eine Funktion, Methode oder Klassendefinition angewendet wird.
- Der Decorator wird angewendet, indem man `@decoratorname` vor die Definition schreibt.
- In diesem Fall wird also der Decorator `profile` auf zu profilierende Funktionen angewendet.
- Das so veränderte Skript lässt sich nicht mehr normal ausführen!

line_profiler Ausgabe

Profiler Ausgabe für die Funktion pseudo_data

```
$ kernprof -l -v tiny_search_engine.py -v 1000 -d 1000
```

```
Timer unit: 1e-06 s

Total time: 3.52772 s
File: tiny_search_engine.py
Function: pseudo_data at line 32

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    32                                           @profile
    33                                           def pseudo_data( l):
    34      1001          456      0.5      0.0      v = []
    35   1002001       316844      0.3      9.0      for i in range(l):
    36   1001000      3210057      3.2     91.0          v.append(random.randint(0,100))
    37      1001          368      0.4      0.0      return v
```

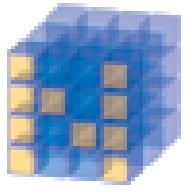
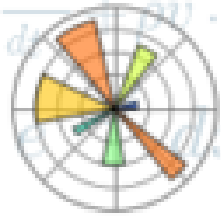
Übung 13

Python für Machine Learning

Scipy

- Sammlung von python-basierter Open Source Software für Mathematik und Naturwissenschaften
- Beinhaltet

NumPy	Numerik-Paket, insbesondere für Vektor- und Matrixrechnung
Scipy	Sammlung von Algorithmen und Werkzeugen
Matplotlib	Paket zum Plotten
IPython	Interaktive Shell und Browserbasierte Notebooks
Weitere	Sympy, Pandas

IP[y]:
IPython

http://www.scipy.org/

Warum Scipy?

- Stellt viele Funktionalitäten zur Verfügung, die Python so nicht anbietet (z.B. Operationen auf Vektoren/Matrizen, höhere Mathematik)
- Schnittstelle zu effizienten C/C++-Libraries
- Warum interessant für Computerlinguisten?
  - Oft Vektorrepräsentation benötigt (Feature-Vektor, Kontextvektor)
  - Große Datenmengen $\rightarrow$ Speicher- und Laufzeiteffizienz notwendig
  - Toolkit für maschinelles Lernen (`scikit-learn`) arbeitet mit SciPy-Funktionen und Datenstrukturen

Ziel für den nächsten Teil: mithilfe von Numpy/Scipy unsere Mini-Suchmaschine optimieren (Geschwindigkeit mit Profiler messen!)

Python für Machine Learning

8 Python für Machine Learning

- Profiling
- Numpy und SciPy
- Die Numpy Package
- Die SciPy-Library
- scikit-learn

NumPy[3]: `ndarrays`

- Wichtigste Daten-Container in Python: *list* und *dictionary*
- NumPy stellt eine weitere Datenstruktur zur Verfügung: den mehrdimensionalen Array `ndarray`
- Unterschied zur Liste: `list` kann verschiedene Arten von Einträgen haben, `ndarray` erlaubt nur Einträge vom selben Typ (integer, floats, strings)
- Beliebig viele Dimensionen möglich

[3] Dieser Teil basiert auf Bressert, Eli: *NumPy and SciPy*. O'Reilly, 2013.

NumPy: ndarray erzeugen

Array kann auf verschiedene Weisen erzeugt werden.

```
import numpy as np

# eindimensionaler Array
myarr1 = np.array([1, 2, 3, 4])

# zweidimensionaler Array
myarr2 = np.array([1, 2, 3],
                  [4, 5, 6])

# 3x4-Array, alle Eintraege sind 0
myarr3 = np.zeros((3, 4))

# Array mit 100K Elementen
myarr4 = np.arange(100000)

# Integer-Array
myarr5 = np.zeros(100, dtype=int)
```

Arrays: Indexing und Slicing

Beispiel (Liste)

```
mylist = [[1, 2], [3, 4]]
elem = mylist[0][1]
```

Beispiel (Array)

```
# Array aus Liste initialisieren
myarr = np.array(mylist)

elem = myarr[0, 1]
column1 = myarr[:,0]
row2 = myarr[1,:]
```

Operationen auf Arrays

Einfache Rechenoperationen werden elementweise ausgeführt.

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([5, 6, 7, 8])
>>> a * b
array([ 5, 12, 21, 32])
>>> b - a
array([4, 4, 4, 4])
>>> a ** 2
array([ 1,  4,  9, 16])
```

NumPy: Lineare Algebra I

Mit Arrays

```
>>> np.dot(a, b) # Skalarprodukt von a und b
70
>>> m = a.reshape((2, 2))
>>> m
array([[1, 2],
       [3, 4]])
>>> m.T  # m transponiert
array([[1, 3],
       [2, 4]])
```

NumPy: Lineare Algebra II

Matrizen

```
>>> m = np.matrix([[1, 2], [3, 4]])
>>> n = np.matrix([[5, 6], [7, 8]])
>>> m * n # Matrix-Multiplikation
matrix([[19, 22],
        [43, 50]])
```

- `matrix`-Objekte können nur 2 Dimensionen haben.
- Auf `matrix`-Objekt wird standardmäßig Matrixmultiplikation ausgeführt, nicht elementweise Multiplikation.

NumPy: Universal Functions (ufunc)

- ufuncs sind mathematische Funktionen, die elementweise auf Arrays angewendet werden:

```
np.add(a,b) # wird auch aufgerufen bei 'a + b'
np.exp(a)
np.sqrt(a)
np.log(a)
```

- Das normale math-Modul kann nicht auf ndarrays angewendet werden.

numpy.random

- Zufallsgenerator für ndarray-Objekte

```
# 2 x 3 Samples von Standardnormalverteilung
>>> np.random.randn(2, 3)
array([[-1.12695503,  0.21605989, -1.73561989],
       [ 0.38781853, -1.98914298, -0.55159214]])
# 10 Integers zwischen 1 und 10
>>> np.random.random_integers(1, 10, 10)
array([ 1,  8,  5,  5,  4,  4, 10,  3,  8,  8])
```

- Viele Wahrscheinlichkeitsverteilungen vorhanden (Multinomialverteilung, Dirichlet-Verteilung, Zipf-Verteilung, ...)

Dokumentation

- NumPy Tutorial:
  `http://wiki.scipy.org/Tentative_NumPy_Tutorial`
- NumPy Befehlsreferenz:
  `http://docs.scipy.org/doc/numpy/reference/`
- `help()`-Funktion und Doc-Strings

Laufzeitvergleich I

Generierung von Pseudodaten mit Python:

```
Total time: 3.73401 s
File: tiny_search_engine.py
Function: pseudo_data at line 32

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    32                                           @profile
    33                                           def pseudo_data(n):
    34      1001          433      0.4      0.0      v = []
    35   1002001       289524      0.3      7.8      for i in range(n):
    36   1001000      3083360      3.1     82.6          freq = random.randint(0, 100)
    37   1001000       360374      0.4      9.7          v.append(freq)
    38      1001          318      0.3      0.0      return v
```

Laufzeitvergleich II

Generierung von Pseudodaten mit NumPy:

```
def pseudo_data_numpy(n):
    v = np.random.random_integers(0, 100, n)
    return v
```

```
Total time: 0.131629 s
File: tiny_search_engine.py
Function: pseudo_data_numpy at line 40

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    40                                           @profile
    41                                           def pseudo_data_numpy(n):
    42      1001       131170    131.0     99.7      v = np.random.random_integers(0, 100, l)
    43      1001          459      0.5      0.3      return v
```

$\rightarrow$ 28 mal schneller!

Python für Machine Learning

Was bietet die SciPy-Library?

- Benutzt NumPy-Arrays für Standardaufgaben aus verschiedenen naturwissenschaftlichen Disziplinen
- Unter anderem:
  - Optimierung (`scipy.optimize`)
  - Statistik (`scipy.stats`)
  - Räumliche Strukturen und Distanzen (`scipy.spatial`)
  - Signal- und Bildverarbeitung (`scipy.signal`, `scipy.ndimage`)
  - Clustering (`scipy.cluster`)
  - Dünn besetzte (sparse) Matrizen (`scipy.sparse`)

Dokumentation

- Scipy Referenz
  `http://docs.scipy.org/doc/scipy/reference/`
- Buch: Bressert, Eli. SciPy and NumPy. O'Reilly, 2012.

Übung 14

Python für Machine Learning

Python für Machine Learning

Was?

- *SciKits* („SciPy Toolkits") sind Add-ons für SciPy
- Für uns interessant: `scikit-learn` - Toolkit für maschinelles Lernen, basierend auf SciPy
- Ausführliche Dokumentation: `http://scikit-learn.org/stable/user_guide.html`

Warum?

- Klare, gut dokumentierte API
- Schnell (verwendet NumPy Strukturen, kompilierten Code, Schnittstellen zu C-Bibliotheken für bestimmte Lernverfahren)
- Robust
- Relativ einfach zu bedienen

Python für Machine Learning

Einige Aufgabentypen I

Klassifikation

Elemente (*items*) sollen anhand ihrer Merkmale (*features*) bestimmten Klassen (*classes*, *labels*) zugeordnet werden. Es ist im Voraus bekannt, welche (und wie viele) Klassen es gibt.

Clustering

Elemente sollen anhand ihrer Merkmale gruppiert werden, so dass ähnliche Dinge in derselben Gruppe sind.

Einige Aufgabentypen II

Structured Prediction

Komplexen Strukturen (z.B. Sätzen) sollen andere komplexe Strukturen (z.B. Parse-Bäume) zugeordnet werden.

Regression

Für reellwertige Eingaben (z.B. Größe einer Wohnung in Quadratmetern) sollen reellwertige Vorhersagen gemacht werden (z.B. potentieller Kaufpreis).

Laufendes Beispiel: Klassifikation

Was sind die Elemente?

- Elemente sind die zu klassifierenden Objekte.
- Im Prinzip alles möglich.
- Wichtig: Nützliche und generalisierbare Merkmale für die Elemente extrahieren!

Was sind die Elemente?

- Elemente sind die zu klassifierenden Objekte.
- Im Prinzip alles möglich.
- Wichtig: Nützliche und generalisierbare Merkmale für die Elemente extrahieren!

Was sind die Elemente?

- Elemente sind die zu klassifierenden Objekte.
- Im Prinzip alles möglich.
- Wichtig: Nützliche und generalisierbare Merkmale für die Elemente extrahieren!

Was sind die Klassen?

- Elemente werden Klassen zugeordnet.
- Am Ende soll jedes Element zu genau einer Klasse gehören[4]
- Einfachste Klassen: “ja” und “nein” (binäre Klassifikation)

[4] Ausnahme: *multi-label classification*

Was sind die Klassen?

- Elemente werden Klassen zugeordnet.
- Am Ende soll jedes Element zu genau einer Klasse gehören[4]
- Einfachste Klassen: “ja” und “nein” (binäre Klassifikation)

[4] Ausnahme: *multi-label classification*

Was sind die Klassen?

- Elemente werden Klassen zugeordnet.
- Am Ende soll jedes Element zu genau einer Klasse gehören[4]
- Einfachste Klassen: “ja” und “nein” (binäre Klassifikation)

[4] Ausnahme: *multi-label classification*

Elemente und Klassen

Beispiele

Problem	Elemente	Klassen
POS-Tagging	Wörter	POS-Tags
Disambiguierung	Wörter im Kontext	Synsets
Anaphernresolution	Anapher/Antezedenten-Paare	Ja/Nein

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Feature-Extraktion

Wichtiger Teil der Arbeit:

- Entscheiden, welche Merkmale verwendet werden sollen
- Code schreiben, der die Merkmale aus den Daten extrahiert

Trainings- und Testdaten

- Für faire und nützliche Ergebnisse darf nicht auf denselben Daten trainiert und getestet werden!
- Prozentuale Aufteilung: 30% der Daten werden zufällig ausgewählt und als Testdaten beiseite gelassen (während des Trainings)
- Cross Validation: Mehrere Trainingsläufe, wobei jeweils ein anderer Teil der Daten als Testset ausgelassen wird. Besonders sinnvoll bei wenig Daten.

Python für Machine Learning

scikit-learn

scikit-learn bietet...

- Überwachte Lernalgorithmen
- Unüberwachte Lernalgorithmen
- Module zur Vektorisierung und Filterung von Daten
- Module zur Validierung und Evaluierung

Verwendung

```
>>> import sklearn
>>> from sklearn import ...
>>> help(sklearn) # Überblick über Pakete
```

Wichtigste Funktionen von Lernalgorithmen

`fit(X[,y])` Lerne von Daten

`predict(X)` Wende Modell auf (Test-)daten an

`transform(X)` Transformiere Daten (z.B. Feature Selection)

Die Funktionen können auch kombiniert werden (z.B. `fit_predict()` oder `fit_transform()`).

Überwachte Methoden I

- Gegeben Beobachtungen X und Klassen Y
- Lerne $f : X \to Y$ so dass eine Zielfunktion minimiert (oder maximiert) wird (`fit()`-Methode)
- Bestimme $y = f(x)$ für ungesehene x (`predict()`-Methode)

`http://scikit-learn.org/stable/supervised_learning.html`

Überwachte Methoden II

Zwei Kategorien

Klassifikation Y besteht aus endlich vielen Elementen (z.B. $Y = \{0, 1\}$)
`sklearn.linear_model.LogisticRegression`,
`sklearn.tree.DecisionTreeClassifier`, ...

Regression Y ist stetig (z.B. $Y = \mathbb{R}$ oder $Y = [0, 1]$)
`sklearn.linear_model.LinearRegression`,
`sklearn.svm.SVR`, ...

Überwachte Methoden: Beispiel

```
>>> from sklearn import linear_model
>>> import numpy
>>> model = linear_model.LogisticRegression()
# erstelle Beispieldaten
>>> X = numpy.array([[0, 1, 1],
...      [1, 0, 1]])
>>> y = [1, 0]
>>> model.fit(X, y)
>>> model.predict(X)
[1, 0]
```

Unüberwachte Methoden I

- Hier sind nur Beobachtungen X gegeben
- Diese sollen z.B. gruppiert werden

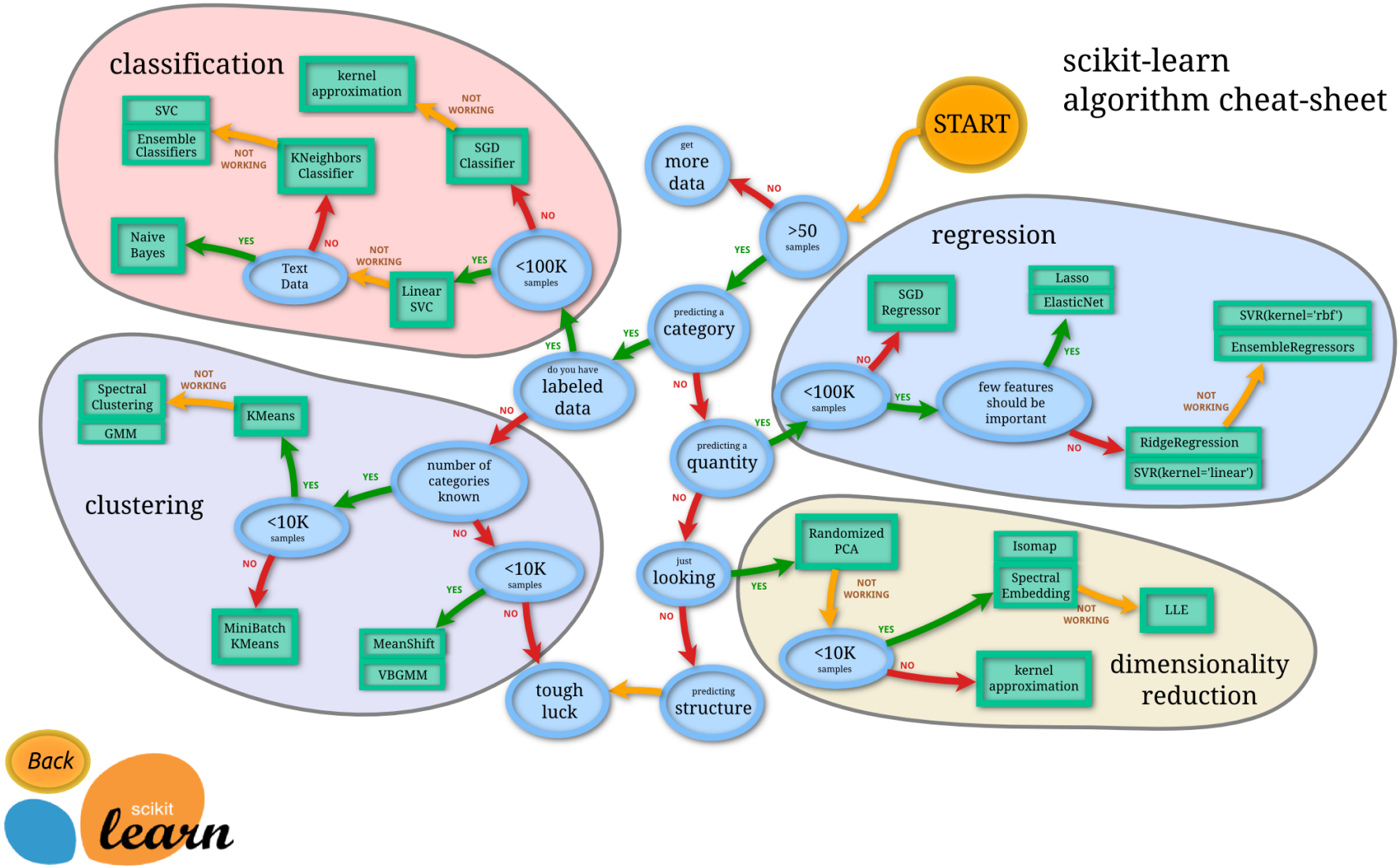
Unüberwachte Methoden II

Einige unüberwachte Lernverfahren

Clustering Daten werden in *Cluster* gruppiert
`sklearn.cluster`

Matrixzerlegung Oft: „Wichtigste" Dimensionen einer Beobachtungsmatrix finden
`sklearn.decomposition`

Welches Lernverfahren wann?



Datenvorbereitung

Preprocessing Normalisierung, Skalierung, kategoriale Features
`sklearn.preprocessing`

Feature-Extraktion Features in korrektes Format für Learner bringen, einfache Text- und Bildfeatures extrahieren
`sklearn.feature_extraction`

`http://scikit-learn.org/stable/modules/preprocessing.html`
`http://scikit-learn.org/stable/modules/feature_extraction.html`

Datenvorbereitung: Beispiel

- `feature_extraction.text` enthält Tools für Umgang mit Textdaten

```
>>> from sklearn.feature_extraction  import text
>>> countvec = text.CountVectorizer()
>>> text = ['Erster Satz', 'Zweiter Satz']
# 'lernt' das Vokabular
>>> countvec.fit(text)
>>> countvec.get_feature_names()
[u'erster', u'satz', u'zweiter']
# 'lernt' das Vokabular und
# generiert Dokument-Term-Matrix
>>> vectors = countvec.fit_transform(text)
# (sparse) Dokument-Term-Matrix
>>> print vectors
  (0, 0)    1
  (0, 1)    1
  (1, 1)    1
  (1, 2)    1
```

Evaluierung

Cross Validation Train/Test Split, Cross Validation (`sklearn.cross_validation`)

Grid Search Hyperparameter einstellen (`sklearn.grid_search`)

Evaluationsmetriken Precision, Recall, F1-Score (`sklearn.metrics`)

`http://scikit-learn.org/stable/model_selection.html`

Evaluierung: Beispiel

```
>>> predicted =  model.predict(X)
>>> sklearn.metrics.accuracy_score(y, predicted)
1.0
```

Modell speichern

- trainiertes Modell (z.B. SVM-Objekt) für spätere Anwendung speichern
- Lösung: Serialisierung mit `pickle` (Python-Modul zur Objekt-Serialisierung)

Übung 15

Mittagspause

Donnerstag: Weka

Weka

Intro

Weka ist ...

- Eine Sammlung von Algorithmen für Machine Learning und Data Mining
- In Java implementiert
- Hat eine GUI und eine API
- Lizensiert unter GNU GPL
- `http://www.cs.waikato.ac.nz/ml/weka/`

Datenformate I

CSV (Comma-Separated Values

- Ein Beispiel pro Zeile
- Merkmale werden durch Komma getrennt

Example

```
Darth, upper, ""
Vader, upper, Darth
war, lower, Vader
ein, lower, war
Lord, upper, ein
der, lower, Lord
Sith, upper, der
...
```

Datenformate II

ARFF - *Attribute Relation File Format*

Standardformat in Weka

Example

```
@RELATION darth-vader
@ATTRIBUTE token STRING
@ATTRIBUTE case {upper,lower}
@ATTRIBUTE previous STRING
@ATTRIBUTE class {name, other}
@DATA
"Darth", upper, "", name
"Vader", upper, "Darth", name
"war", lower, "Vader", other
"ein", lower, "war", other
...
```

Datenformate III

Syntax von ARFF

- `@RELATION name`
  definiert einen Namen für das Datenset
- `@ATTRIBUTE attribute TYPE`
  definiert ein Attribut namens "attribute" vom Typ TYPE

string	Zeichenketten
numeric, real, integer	Zahlen
{ nom1, nom2 }	Listen nominaler Werte
date	Datumsangaben (yyyy-MM-dd'T'HH:mm:ss)

- `@DATA`
  Hier stehen die einzelnen Elemente (in CSV-Format)

Datenformate IV

Beispiel nominaler Werte

- { red, green, blue }
- { gabi, paula, anna-katharina }
- { one, two, three }
- { true, false }

Konvertierung

Sind alle Zeichenketten in einem Datenset bekannt, können sie automatisch in nominale Werte konvertiert werden.

Datenformate V

Annotation, fehlende Werte, Sonderzeichen

- Klassen werden im Attribut *class* angegeben, normalerweise als letztes Attribut
- Fehlende Werte werden mit einem `?` gekennzeichnet.
- Kommentare beginnen mit '%'
- Sonderzeichen (z.B. '?',',', '%') müssen in Anführungszeichen stehen, wenn sie nicht in ihrer Sonderbedeutung vorkommen.

Weka Benutzeroberfläche

Weka GUI Chooser

- ausführen mit

```
:~$ java -jar /path/to/weka.jar
```

- **Explorer**: Daten importieren, bearbeiten und visualisieren
- **Experimenter**: Experimente mit unterschiedlichen Parametern durchführen
- **KnowledgeFlow**: Komponenten und Datenströme graphisch modellieren
- **Simple CLI**: Command Line Interface

CLI

- Jeder Klassifizierer ist in einer Java-Klasse implementiert
- Aufruf über die Kommandozeile möglich:[5]

```
:~$ java weka.classifiers.trees.J48 <parameter>
```

- Parameter: Manche Parameter werden von jedem Klassifizierer verwendet (zum Beispiel Angabe der Trainings- und Testdaten), manche Parameter sind spezifisch für bestimmte Klassifizierer
- Wird der Klassifizierer ohne Argumente gestartet, zeigt der help screen alle Parameter an

[5] J48: Weka-Implementierung eines Entscheidungsbaums

Weitere Informationen zu Weka

- Witten, Frank & Hall (2011): Data Mining. Morgan Kaufman.
  $\rightarrow$ UB
- Online-Kurse: `https://weka.waikato.ac`

Übung 16

Freitag: Arbeiten auf dem Cluster

10 Arbeiten auf dem Cluster

Arbeiten auf dem Cluster

Arbeiten auf dem Cluster

- Nicht alle Prozesse können auf den Servern oder privaten Rechnern ausgeführt werden
- Parallele Verarbeitung von großen Daten auf vielen Rechnern
- Bedarf zusätzlicher Ressourcen
  - (Mehrere) Grafikkarten
  - Große Arbeitsspeicher
  - Mehrere Prozessoren

Cluster

- Cluster: Ansammlung von Rechner, die gemeinsam Rechenressourcen zur Verfügung stellen
- Das ICL hat ein Cluster aus 54 Maschinen
  - 45 Rechner für normale compute-Aufgaben (ohne GPU)
  - 9 GPU Rechner mit bis zu 8 GPUs
- Das Cluster ist die einzige Variante, auf die Instituts-CPUs zuzugreifen
- Wichtige Mailingliste: `http://lists.cl.uni-heidelberg.de:8080/mailman/listinfo/cluster-users`
- **Viele Infos im Wiki** `https://wiki.cl.uni-heidelberg.de/bin/view/Main/FaQ/Tutorials/SlurmTutorial`

Anmeldung

```
:~$ ssh cluster
```

- Zugriff muss von der Technik freigeschaltet werden
- Loggt sich auf Verwaltungsknoten ein $\rightarrow$ Hier keine Aufwändigen Tasks starten

Arbeiten auf dem Cluster

Cluster Ressourcenallocation

- Das Cluster wird von vielen Menschen gleichzeitig genutzt
- Effiziente Ressourcenallokation entscheidend
  - Welche Prozesse laufen wo?
  - Wie viel Speicher bekommt der Prozess?
  - Wer braucht Zugriff auf exlusive Ressourcen Grafikkarten)?
- ICL-Cluster verwendet Slurm zur Verwaltung

Slurm

- *Simple Linux Utility for Resource Management*
- Sog. Workload Manager, verwaltet Clusterressourcen
- Terminologie

Node	Ein einzelner Computer
Partition	Gruppierung von Computern (zu Verwaltungszwecken)
Job	Ansammlung von Ressourcen (CPUs, Speicher, ...), die einem Benutzer zur Ausführung seiner Programme freigegeben wurden
Job Step	Einzelner Prozess, der als Teil eines Jobs ausgeführt wird

- Dokumentation: `https://slurm.schedmd.com`

Slurm - Ablauf

1. Benutzer bittet Slurm um bestimmte Ressourcen aus einer Partition
2. Ressourcen werden dem Benutzer zugewiesen (man sagt *alloziert*)
   - Jede Partition hat eine Warteschlange (queue)
   - Wenn die Ressourcen belegt sind, müssen wir warten, bis sie frei sind
3. Benutzer kann einzelne Job-Steps starten
   - Steps können Ressourcen belegen, die uns alloziert wurden
   - Wir können mehrere Steps nebeneinander ausführen
4. Nach einem Zeitlimit, oder nach Ende aller Schritte, werden die Ressourcen freigegeben

Arbeiten auf dem Cluster

Slurm bedienen

- Slurm Commands liegen in `/opt/slurm/bin`
- Sollten in die Pfadvariable gelegt werden
- Einfachste Variante: Befehl in `~/.profile` schreiben
  - `~/.profile` wird bei Login ausgeführt

```
if [ -d "/opt/slurm/bin" ] ; then
    PATH="/opt/slurm/bin:$PATH"
fi
```

Slurm bedienen - Partition wählen

- Zu Beginn müssen wir die passende Partition wählen
- Ergibt sich aus unseren Anforderung
- Unterschiedliche Partition haben verschiedene Nodes und Timelimits

Slurm bedienen - Partition wählen

Name	Zeitlimit	Zweck
main	3 Tage	30 Ältere CPU-Nodes
compute	3 Tage	15 Moderne CPU-Nodes
gpushort	8 Stunden	GPU-Nodes für kurze Tests
gpulong	3 Tage	GPU-Nodes
students	3 Tage	GPU für Studierende

Slurm bedienen - Infos einholen

- Bevor wir anfangen, sollten wir uns den Zustand des Clusters anschauen

  sinfo Zeigt Infos über Cluster-Knoten an
  - Welche Nodes gibt es in welchen Partitionen?
  - Sind Nodes nicht verfügbar (down)?
  - Welche Nodes werden benutzt?

  squeue Zeigt Infos über laufende und wartende Jobs an
  - Zeigt Job-Status in Spalte `ST` an
  - Zentrale Statuscodes: `R` für laufende Jobs, `PD` (pending) für Jobs, die auf Ressourcen warten
  - Sind viele Jobs für eine Partition pending, werden wir potenziell lange warten müssen

Interaktiver Modus

- Wir können auf dem Cluster (fast) wie auf einem lokalen Computer arbeiten
- Das nennen wir "Interaktiven Modus"
- Das besteht aus zwei Schritten:
  1. `salloc` alloziert Ressourcen
  2. `srun` führt Kommandos aus (quasi wie Bash)
- Geeignet für Konfiguration und Tests

Interaktiver Modus - `salloc`

- `salloc` benötigt eine Spezifikation unserer Anforderung an das Cluster
- `-p <partition>` wählt Partition aus
- `--time hh:mm:ss` legt maximale Laufzeit fest
- `--cpus-per-task <n>` legt Anzahl der CPUs fest, die pro Task alloziert weden
- `-N<n>` legt minimale Anzahl der Nodes fest, die reserviert werden
- `-n<n>` legt Anzahl der Prozessoren fest, die wir brauchen
- `--gres=gpu:<n>` reserviert *n* GPUs
- Wartet, bis Anforderung Erfolg hatte

Interaktiver Modus - `srun`

- Nachdem `salloc` fertig ist, können wir mit srun Befehle ausführen

```
:~$ srun <command>
```

- `-n` erlaubt festzulegen, wie Instanzen des Programms gleichzeitig ausgeführt werden (Normalerweise benötigen wir nur eine Instanz)
- Achtung: Wir befinden uns weiterhin auf unserem ursprünglichen (Cluster-Login-)Computer, erst `srun` startet Befehle auf dem Cluster

Interaktiver Modus - `scancel`

- Nachdem wir fertig sind, sollten wir unsere Ressourcen freigeben

```
:~$ scancel <job-id>
```

- Job-ID wird von `salloc` zurückgegeben
- Wenn vergessen: `sacct` listet Jobs
  - Listet unsere ausgeführten Jobs
  - Laufende Jobs haben den Status `RUNNING`
  - Job-ID ist in der ersten Spalte

Batch Modus

- Interaktiver Modus ist nur zum Testen da
- Für den Produktiveinsatz: `sbatch`
- Erwartet als Eingabe Batch-file

Batch-Files

- Beschreibt einen Job für das Cluster
- Besteht aus Befehlen und `#SBATCH`-Direktiven
  - `#SBATCH` beschreibt `salloc`-Parameter
  - Befehle werden wie ein (Bash-)Script ausgeführt
- `#SBATCH` Direktiven können einfach von salloc übernommen werden
- Ausgabe wandert per default in `slurm-<jobid>.out`

Batch-Files - Beispiel

```
#!/bin/bash
#SBATCH --job-name=my_long_batch_job
#SBATCH --output=long_batch_result.txt
#SBATCH --mail-user=YOUR_USERNAME@cl.uni-heidelberg.de
#SBATCH --mail-type=ALL
#SBATCH --partition=compute
#SBATCH --time=1-0:00:00

# JOB STEPS
srun hostname # example job step
srun echo $CUDA_VISIBLE_DEVICES # another example job st
```

siehe Wiki

Warum Batch-Files?

- Beenden automatisch, sobald alle Schritte ausgeführt wurden
- Befehle werden ausgeführt, sobald Ressourcen alloziert wurden
- Fortschritt kann per Mail an uns gesendet werden

```
#SBATCH    --mail-type=ALL
#SBATCH    --mail-user=<USERNAME>@cl.uni-heidelberg.de
```

Arbeiten auf dem Cluster

Daten auf dem Cluster

- Cluster hat kein Ressourcen Verzeichnis
- Home-Verzeichnisse auf dem Cluster werden nicht mit Servern/Rechnern synchronisiert
- Daten müssen manuell transferiert werden

Werkzeuge zum Datentransport

- `scp <src-host>:<src> <dest-host>:<dest>`
  - Kopiert Datei `src` auf `src-host` nach `dest` auf `<dest-host>`
  - Wenn `src` sich auf dem lokalen Rechner befindet, kann `src-host` weggelassen werden (genauso für `dest-host`)
- `rsync`
  - Grundlegende Nutzung wie `scp`
  - Erkennt, wenn Daten im Ziel schon vorhanden sind
  - Oft schneller

Daten löschen

- Auf dem Cluster ist nicht unbegrenzt viel Platz
- Daten sollten gelöscht werden, wenn man sie nicht mehr braucht

Arbeiten auf dem Cluster

Slurm Array Jobs I

- Cluster ist nicht nur gut, um GPUs zu benutzen
- Nützlich, wenn man viele Inputdaten im selben Format hat, die auf gleiche Art und Weise verarbeitet werden
- Beispiel: Parsing von großen Corpora
- Slurm hat einen eingebauten Mechanismus: Job Arrays

Slurm Array Jobs I

```
:~$ sbatch --array <start-id>-<end-id> <script>
```

- Benötigt Batch-Script
- Startet übergebenes Batch-Script einmal für jede Zahl zwischen start-id und end-id
- Umgebungsvariable `$SLURM_ARRAY_TASK_ID` wird auf die momentane ID gesetzt

Beispiel I

- Wir nehmen an, wir haben einen Corpus in den Dateien `corpus-1`, `corpus-2`, ..., `corpus-20`
- Fiktives Programm `parse` soll Dokumente verarbeiten
- Jeder Split wird unabhängig von den anderen bearbeitet

Batch-Datei

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --array=1-20

srun ./parse corpus-$SLURM_ARRAY_TASK_ID
```

Arbeiten auf dem Cluster

Virtualenv

- Cluster stellt `python 2 + 3` zur Verfügung
- Bibliotheksauswahl ist aber begrenzt
- Lösung: Virtual Environment
  - Kapselt packages in lokalem Verzeichnis
  - Isoliert Installation vom Rest des Systems

Installation und Aktivierung

- Python2:

```
:~$ virtualenv --system-site-packages <env-name>
```

- Python3:

```
:~$ python3 -m venv --without-pip <env-name>
```

- oder

```
:~$ virtualenv -p python3 --system-site-packages <
env-name>
```

- Anschließend:

```
:~$ source <env-name>/bin/activate
```

- Neue Pakete installieren (nach Aktivierung):

```
:~$ pip install <package-name>
```

Python3 venv

- `--without-pip` ist ein Workaround für problematisches Debian Verhalten
- Nach Installation fehlt `pip` in der Environment
- Lösung: **Nach** Aktivierung `pip` installieren

```
:~$ curl https://bootstrap.pypa.io/get-pip.py | python
```

Übung 17

Arbeiten auf dem Cluster

GPU -CUDA I

- Installationsprozeduren für verschiedene GPU-Werkzeuge sind unterschiedlich
- Meistens müssen wir CUDA einbinden
  - Werkzeug-Kasten für Programmierung auf GPU
- Dafür müssen wir ein paar Variablen richtig setzen
  - Wiki enthält passendes Script
  - `https://wiki.cl.uni-heidelberg.de/bin/view/Main/FaQ/Tutorials/GpuHowto`

GPU - Bibliotheken I

- Da CUDA global installiert ist, müssen wir die passenden Versionen unserer Lieblingsbibliothek suchen, die darauf passt
- `https://wiki.cl.uni-heidelberg.de/bin/view/Main/FaQ/Tutorials/GpuHowto` gibt Hinweise
- Python-Bibliotheken müssen in Virtual Environment installiert werden
- Um GPUs nutzen zu können, müssen sie angefragt werden

```
#SBATCH --gres gpu:1
```

Cluster Etikette I

- Cluster wird von vielen Menschen benutzt
- Vielen brauchen schnelle Ergebnisse
- Viele Nutzer sorgen für schnelle Verstopfung und lange Queues
- Deswegen sollte man die Ressourcen so wenig wie möglich beanspruchen (v.a. Grafikkarten)

Cluster Etikette II

- Jobs beenden, sobald fertig (Batch-Files)
- Nicht mehr Ressourcen allozieren, als man braucht
- Ressourcenschonender Arbeitsablauf
  1. Erst lokal testen (z.B. ohne GPU) und debuggen
  2. Dann erst z.B. auf students ausführen