

# Practical Speech Processing

---

```
svn checkin einkaufszettel.txt
```

---

# Programm für heute

---

- Referats- und Projektideen, Referatstermine!
  - Subversion en gros
  - Subversion anwenden
  - Depots anlegen
  - Tags und Branches
-

# Referate

---

- das erste Referat ist schon in zwei Wochen
- wer hält wann sein Referat?



# Subversion ...

---

- verwaltet Daten in einem zentralen **Depot**
    - Dateien, Verzeichnisse, Meta-Daten
  - ist eine Zeitmaschine
    - Änderungen der Daten werden im **Verlauf** gespeichert
    - Änderungen lassen sich bequem zurücknehmen
  - unterstützt **verteiltes Arbeiten** (allein oder miteinander), durch mehrere, gleichzeitige **Arbeitskopien**
-

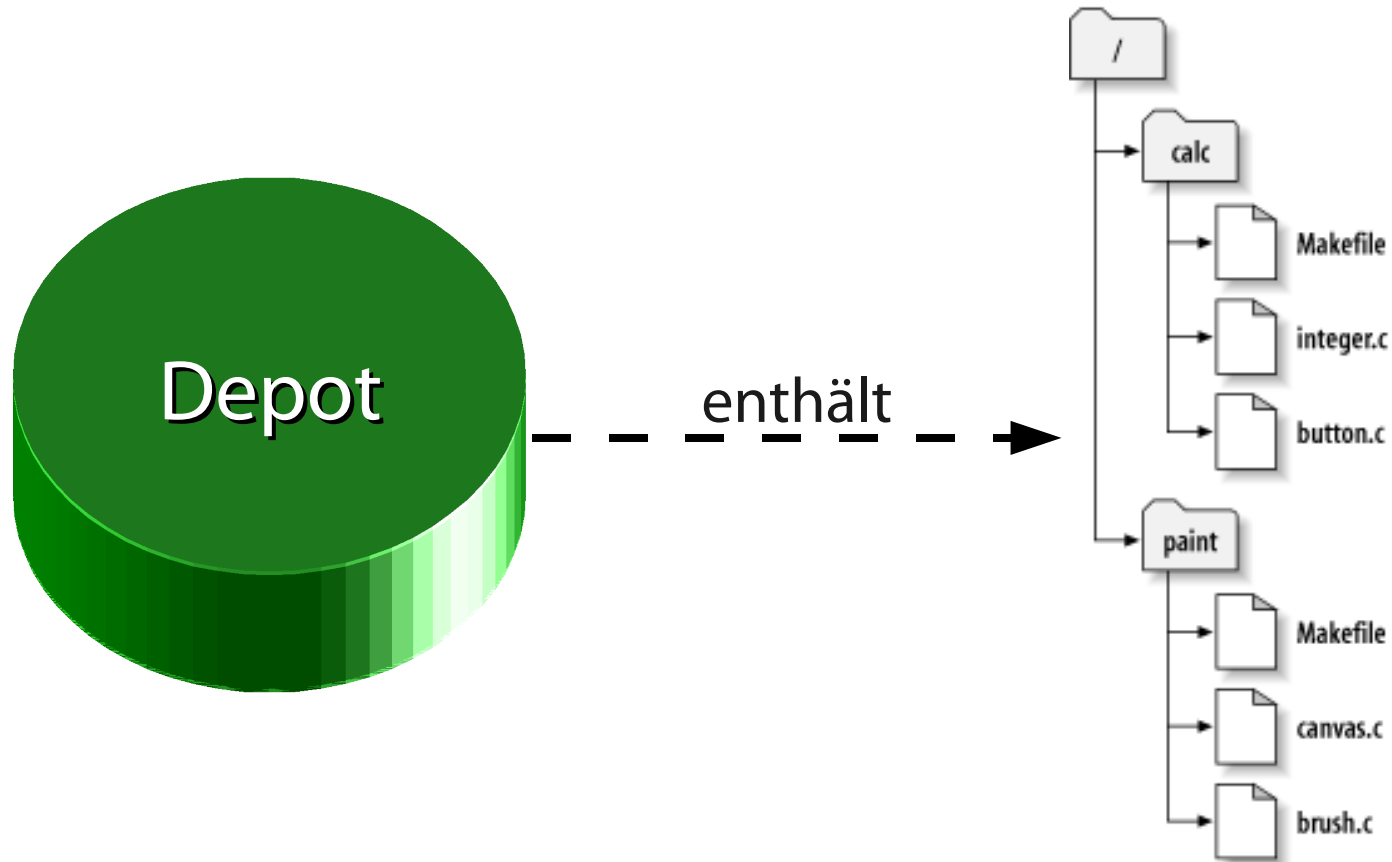
# Depot (engl. Repository) (I)

---

- enthält ein „virtuelles Dateisystem“
    - in einer Datenbank
  - dem Repository werden Dateien und Verzeichnisse hinzugefügt, gelöscht, verändert, kopiert, ...
  - alle Veränderungen werden mitgespeichert
  - wird (nur!) über Subversion-Tools benutzt
  - liegt zum Beispiel auf einem Server
-

# Depot und virtuelles Dateisystem

---



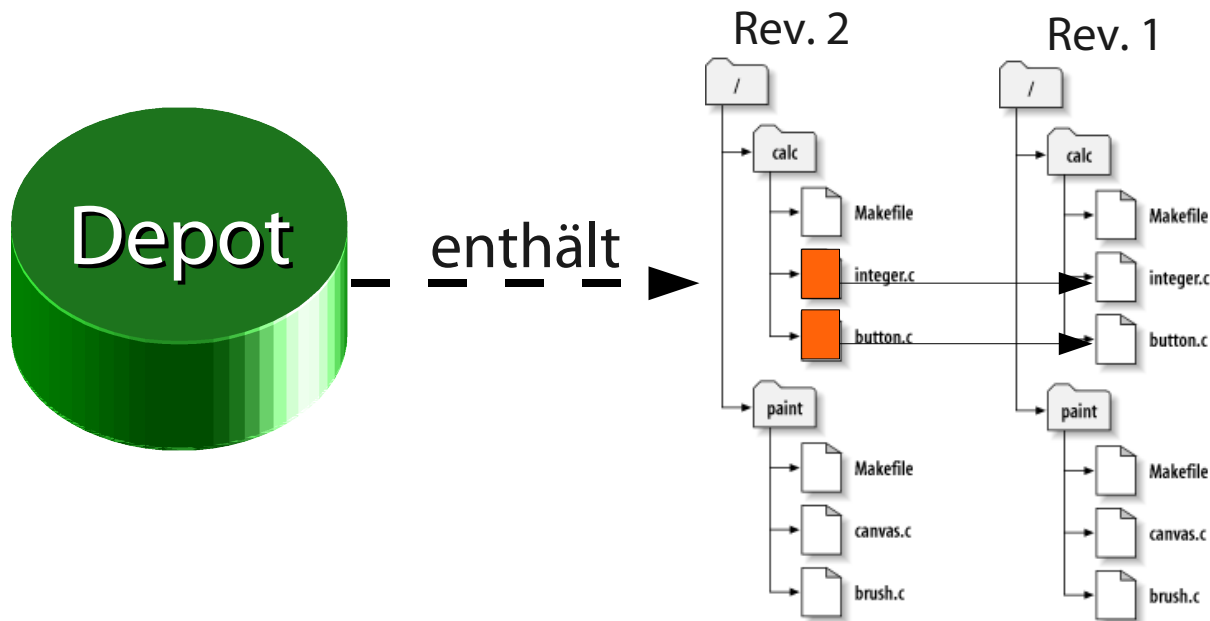
# Depot (engl. Repository) (II)

---

- jede Veränderung am Depot führt zu einer **Revision** des virtuellen Dateisystems
    - Revisionen werden durchnummeriert
    - auf jede Revision des Dateisystems ist zugreifbar
      - ♦ und damit alle Änderungen an Dateien und Verzeichnissen
  - es werden jeweils nur die Änderungen gespeichert, dadurch bleibt der Speicherbedarf moderat
-

# Depot mit mehreren Revisionen

---



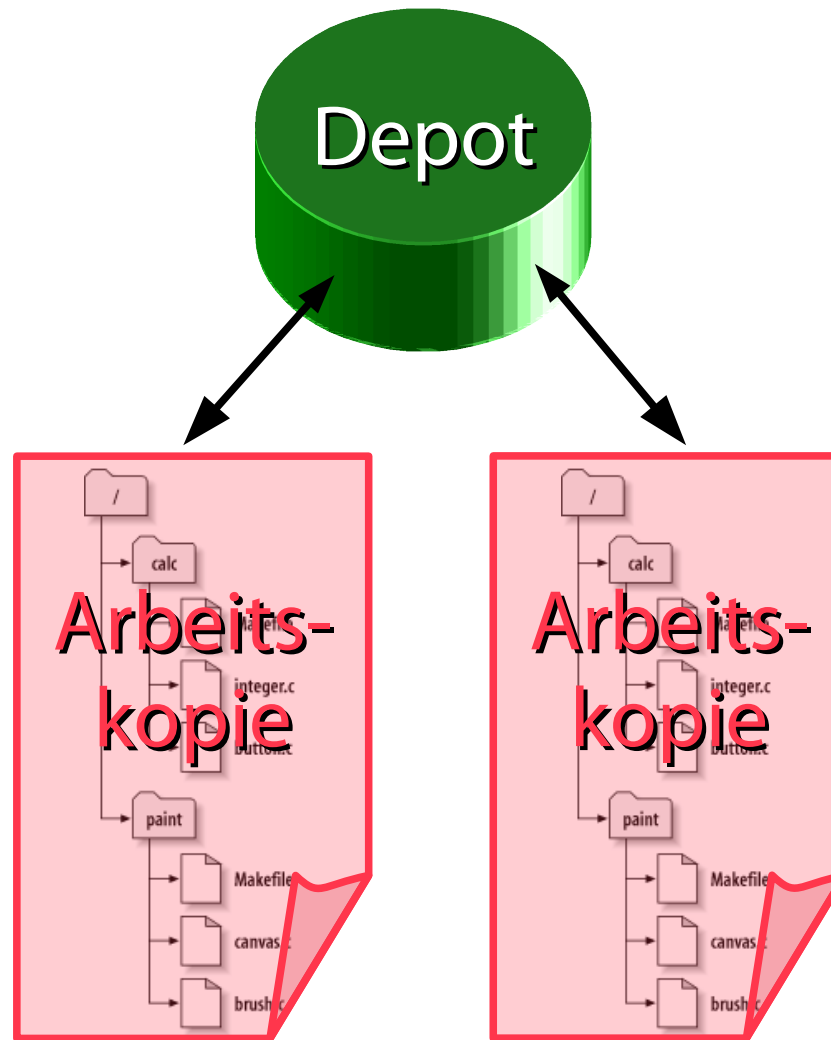
# Arbeitskopie (I)

---

- enthält die Daten des „virtuellen Dateisystems“
    - oder einen Verzeichnisast daraus
  - hier werden Änderungen vorgenommen
    - und anschließend in das Depot übertragen
  - es kann beliebig viele Arbeitskopien zugleich geben
    - Depot: Server, Arbeitskopie(n): Client(s)
  - jede Arbeitskopie merkt sich, zu welchem Repository sie gehört
-

# Depot und Arbeitskopien

---



# Arbeitskopie (II)

---

- normaler Verzeichnisbaum
    - zusätzliche Verwaltungsdaten in `.svn/`-Verzeichnissen
  - in der Arbeitskopie können auch nicht-versionierte Dateien stehen
  - Dateien müssen explizit der Versionierung hinzugefügt werden!
    - `svn add neue.datei`
-

# Verbindung zu Make (letzte Woche)

---

- nur Quelldateien und Makefile zur Versionierung hinzufügen
  - abhängige Dateien macht dann `make all` in der Arbeitskopie



# taglich Brot – svn Kommandos (I)

---

- `svn help` – wie ging das noch?
    - `svn help checkout`, `svn help commit`, `update`, ...
  - am Anfang wird eine Arbeitskopie aus dem Repository „ausgecheckt“
    - `svn checkout file:///home/timo/SVN/psp08/`
  - Repository-URLs:
    - fur lokales Repository (`file:///Pfad/zum/Repository/`)
    - uber SSH: (`svn+ssh:///server.name/Pfad/zum/Repo/`)
-

# Beispiel

---

- checkt mal `svn+ssh://helios.ling.uni-potsdam.de/home/timo/SVN/psp08/` aus
  - man kann auch (ohne auszuchecken) den Inhalt eines Depots anschauen:
    - `svn info URL` (Informationen über ein Repository)
    - `svn list URL` (Verzeichnislisting)
    - `svn cat URL` (Dateiausgabe)
-

# Änderungen vornehmen

---

- Änderungen an Dateien bemerkt Subversion selbst
  - Dateien löschen, verschieben, kopieren:
    - `svn remove`, `move`, `copy`
    - `svn mkdir` – für versionierte Unterverzeichnisse
  - Dateien der Versionierung hinzufügen
    - `svn add`
-

# Änderungen übertragen

---

- führt zu neuer Revision im Repository
  - Änderungen an einer oder mehrerer Dateien werden gemeinsam committet
    - möglichst so, dass es logisch zusammengehört
  - Jedes Commit wird mit einem Logeintrag versehen
  - jedes Commit ist atomar, d. h. entweder alle oder keine Veränderung sind nachher gespeichert
  - evtl. muss zunächst ein Update ausgeführt werden
-

# Updates: Änderungen einbeziehen

---

- Commit führt nicht automatisch zu Update
  - Update versucht automatisch, kompatible Änderungen vorzunehmen
    - lokal unverändert: übernehmen
    - lokal verändert: automatisch verbinden?
      - ♦ funktioniert (nur) bei textbasierten Dateiformaten
      - ♦ sonst durch Nutzer manuell mergen lassen
-

# taglich Brot – svn Kommandos (II)

---

- lokal was andern: Datei erstellen und hinzufugen
  - lokale anderungen anzeigen
    - **svn status**
  - anderungen und Hinzufugungen mussen explizit zum Repository ubermittelt werden
    - **svn commit -m „message“**
  - Repository-anderungen von anderen einspielen:
    - **svn update**
-

# Beispiel

---

- lege in der Arbeitskopie im Unterverzeichnis subdir/ eine Datei an, übertrage sie, ändere sie und so weiter.
  - sprich Dich mit Deinem Nachbar ab und ändert gegenseitig Eure Dateien – funktioniert das automatische mergen?
-

# taglich Brot – svn Kommandos (III)

---

- nderungen rckgngig machen
  - **svn revert** – zur letzten Revision
- ltere nderungen knnen auch wiederhergestellt werden mit **svn revert -r 123**



# Arbeit „taggen“ und „branchen“

---

- Subversion nutzt kein explizites Tagging (wie CVS)
  - stattdessen wird eine Revision in ein bestimmtes Verzeichnis (z. B. tags/) kopiert
    - dies braucht im Repository kaum zusätzlichen Speicher
  - später kann dann das Verzeichnis aus dem Repository ausgecheckt werden um den Arbeitsstand abzurufen
  - Projektverzweigungen funktionieren genauso
-

# Repositoryys anlegen

---

- **svnadmin create <Pfad>**
    - legt ein neues Repository an
    - zusätzlich Parameter sind üblicherweise egal
  - **svn import <Pfad> <URL>**
    - den Inhalt eines Verzeichnisses in ein (noch leeres) Repository importieren
    - das Verzeichnis wird dadurch natürlich noch keine Arbeitskopie! (dafür dann svn checkout)
-

# Merging

---

- copy-modify-merge
-

# Gemeinsam arbeiten

---

- Commits können von diversen Arbeitskopien kommen
  - falls notwendig, muss zunächst ein update durchgeführt werden (das sagt svn von selbst)
  - im gemeinsamen Repository liegt die Datei `index.html`, tragt bitte Euren Namen und Euer Thema beim richtigen Termin ein und checkt die fertige Datei ein.
-

# komfortabel: svn in Eclipse

---

- Subclipse
  - Plugin, das die Subversionbenutzung weitgehend in Eclipse integriert und automatisiert

# Unterschiede zu CVS

---

- Verzeichnisse werden genauso versioniert wie Dateien
    - bessere Unterstützung für kopieren, verschieben, löschen, ...
  - versionierte Metadaten (Attribut-Wert-Paare) zu Dateien und Verzeichnissen
  - konsistentere interne Datenhaltung
  - effizientes Branching und Tagging
-

bazaar, mercurial

---

---