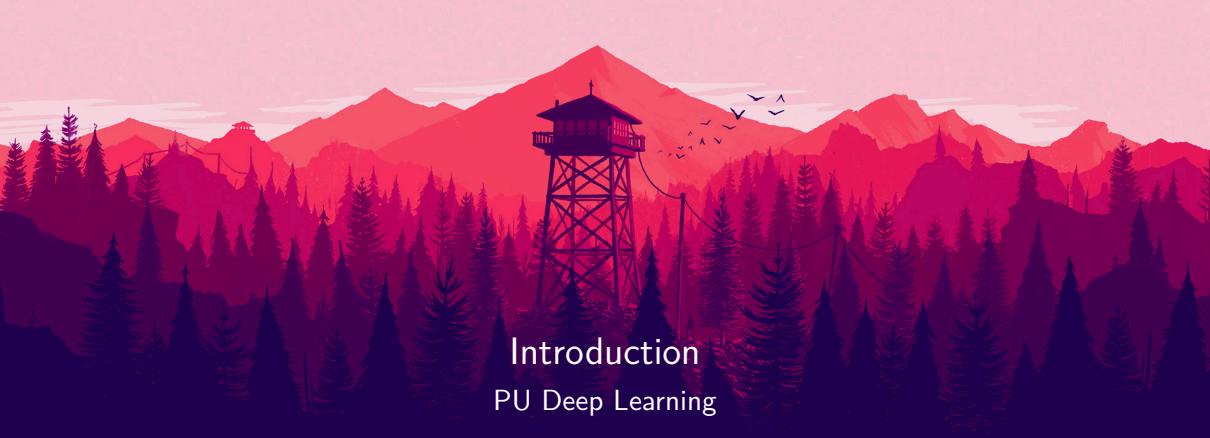


Introduction

PU Deep Learning

Nils Reiter, nils.reiter@uni-koeln.de

April 13, 2021 (Summer term 2021)

Deep Learning

Multiple, overlapping trends (›deep learning‹ is a buzz word)

Deep Learning

Multiple, overlapping trends (›deep learning‹ is a buzz word)

- Distributed representations: Input instances as vectors
  - No ›manual‹ feature extraction
  - Instances are ›embedded‹ into high-dimensional vector space
    - Distributed: Embedding incorporates information from contexts of multiple instances
  - More training data
  - Embeddings are learned from data, therefore: ›representation learning‹

Deep Learning

Multiple, overlapping trends (›deep learning‹ is a buzz word)

- Distributed representations: Input instances as vectors
  - No ›manual‹ feature extraction
  - Instances are ›embedded‹ into high-dimensional vector space
    - Distributed: Embedding incorporates information from contexts of multiple instances
  - More training data
  - Embeddings are learned from data, therefore: ›representation learning‹
- (Artificial) Neural networks: Classification algorithms (first described in the 1940s!)
  - Neuron: Building block
  - Takes input from previous neurons, applies mathematical function, outputs to next function

Python

- ›New‹ programming language (1.0: 1991)
- General-purpose, high-level
- Popular for machine learning and natural language processing
- Almost completed transition between 2.x and 3.x

Python

- ›New‹ programming language (1.0: 1991)
- General-purpose, high-level
- Popular for machine learning and natural language processing
- Almost completed transition between 2.x and 3.x
- Differences to Java/C++
  - Dynamic typing
  - Indentation part of syntax
  - Interpreted language

Nils Reiter

- Vertretungsprofessor Sprachliche Informationsverarbeitung / Digital Humanities (seit September 2019)
- Davor: Uni Stuttgart, Uni Heidelberg, Uni Saarbrücken
- Studium Computerlinguistik / Informatik
- Forschungsinteressen
  - Angewandte Sprachtechnologie
  - Computational Literary Studies
  - Formalisierung / Operationalisierung komplexer Probleme

Section 2

Formalitäten und Ablauf

Kursablauf

Lernziele

- Code mit git versionieren
- Python-Code lesen und schreiben
- Deep-Learning-Experimente mit tensorflow durchführen
- Verschiedene DL-Architekturen (auch konzeptuell) kennen: *recurrent*, *convolutional*, *long-short-term memory* neural networks

Kursablauf

Lernziele

- Code mit git versionieren
- Python-Code lesen und schreiben
- Deep-Learning-Experimente mit tensorflow durchführen
- Verschiedene DL-Architekturen (auch konzeptuell) kennen: *recurrent*, *convolutional*, *long-short-term memory* neural networks

Studienleistung

- Jede Woche eine Übung committen
- Stand ist in Ilias einsehbar

Kursablauf

In jeder Sitzung:

- Einführung in neues Thema
- Vorstellung der Übungsaufgabe
- Lab session: Arbeit in Kleingruppen an der Übung
- Fertigstellen der Übung bis Montagabend der nächsten Woche
  - Abgabe der Übungen in einem eigenen branch auf via GitHub
- Kommentierte Referenzlösung erscheint als update via GitHub

Kursorganisation

Ressourcen, Literatur, Kommunikation

- Kurswebseite
  `https://lehre.idh.uni-koeln.de/lehrveranstaltungen/sosem21/deep-learning/`
  - Folien, Zeitplan (wird ggf. aktualisiert)
- GitHub-Gruppe: `https://github.com/idh-cologne-deep-learning`
  - Übungen erscheinen dort als Repository
  - Manchmal gibt es ein Update für ein existierendes Repository

Kursorganisation

Ressourcen, Literatur, Kommunikation

- Kurswebseite
  `https://lehre.idh.uni-koeln.de/lehrveranstaltungen/sosem21/deep-learning/`
  - Folien, Zeitplan (wird ggf. aktualisiert)
- GitHub-Gruppe: `https://github.com/idh-cologne-deep-learning`
  - Übungen erscheinen dort als Repository
  - Manchmal gibt es ein Update für ein existierendes Repository
- Discord
  - Sitzungen
  - Server kann auch außerhalb der Sitzungen benutzt werden
- E-Mail `nils.reiter@uni-koeln.de`
  - Alles andere

Modulprüfung: Angewandte Linguistische Datenverarbeitung

Ilias: BA-AM1-Angewandte-Linguistische-Datenverarbeitung.pdf

- Thema
  - Findung und Wahl: Ihre Aufgabe
  - Kann, muss aber nicht, etwas mit dem Seminar zu tun haben
  - Mit mir absprechen
- Praktischer Anteil: Offen.
  Beispiele: Experiment zur automatischen Identifikation eines Textphänomens, Annotationsexperiment, quantitativer Vergleich verschiedener Korpora, ...
- Am Ende: Hausarbeit von ca. 10 S Länge
- ›Letzte‹ Übung vor der Bachelor-Arbeit
- Reden über Ideen für Modulprüfungsthemen am 22.06. (im Hauptseminar)

Section 3

Version Control

Version control

- Versioning of source code
- Differences between versions
- Maintaining several branches in parallel

Version control

- Versioning of source code
- Differences between versions
- Maintaining several branches in parallel

Why is this useful?

- Programming projects quickly become massive
  - Windows 2000: 28mio LoC (ca. 930k standard pages)
  - CorefAnnotator: 27k LoC (ca. 770 standard pages)

Version control

- Versioning of source code
- Differences between versions
- Maintaining several branches in parallel

Why is this useful?

- Programming projects quickly become massive
  - Windows 2000: 28mio LoC (ca. 930k standard pages)
  - CorefAnnotator: 27k LoC (ca. 770 standard pages)
- Large teams
  - working on the same project
  - over a long time (don't rely on human memory)

Version control

- Versioning of source code
- Differences between versions
- Maintaining several branches in parallel

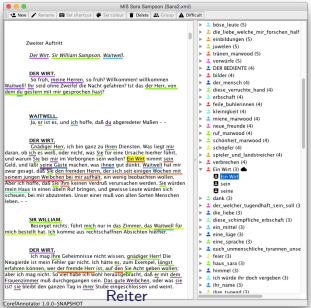
Why is this useful?

- Programming projects quickly become massive
  - Windows 2000: 28mio LoC (ca. 930k standard pages)
  - CorefAnnotator: 27k LoC (ca. 770 standard pages)
- Large teams
  - working on the same project
  - over a long time (don't rely on human memory)
- A single conceptual change often distributed over many files

CorefAnnotator

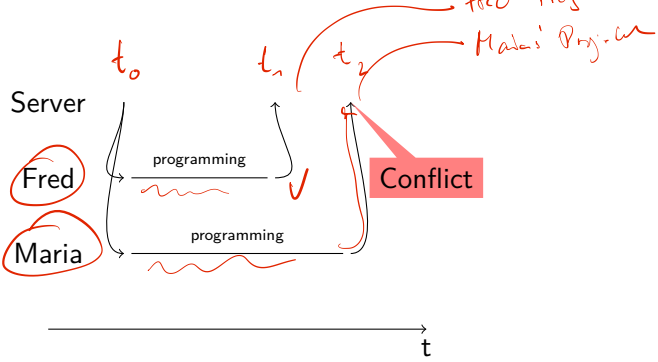
- Typical research software
- `https://github.com/nilsreiter/CorefAnnotator/`
- Annotation tool for coreference chains
- Open source, Apache License
- Version 1.0: February 12, 2018

Version	LoC
1.0	13 087
1.1	14 028
1.2	15 781
1.3	18 988
1.4	20 742
1.5	22 034
1.6	22 792
1.7	22 967
1.8	26 225
1.9.2	27 052



Reiter

Situations



Conflict resolution options

- Ignore, let Maria overwrite Freds code (this is bad!)
- Create a second copy (this is what Dropbox does)
- Force Maria to *explicitly* merge the code

What do we put under version control?

plain text files

- source code (python, java, perl, c, ...)
- texts (plain, latex, markdown)
- primary data (xml, csv)
  - but beware of large files
- vector graphics (svg) (xml)

What do we put under version control?

plain text files

- source code (python, java, perl, c, ...)
- texts (plain, latex, markdown)
- primary data (xml, csv)
  - but beware of large files
- vector graphics (svg)

Don't put these in VC:

Binary files

- word documents, pdf files
- images (jpg, png)
- compiled code (executables)

Software

- Very old
  - CVS (concurrent versioning system)
  - Rarely used today
- Old
  - SVN (subversion)
  - Sometimes used
- State of the art
  - git
- More solutions are available commercially

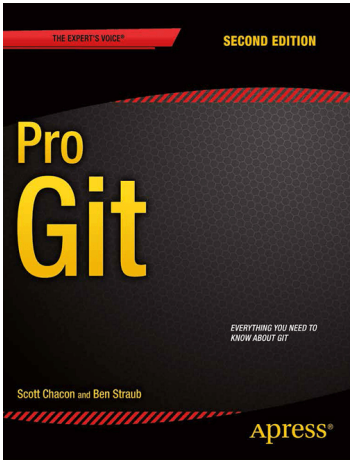
git

- Developed by the Linux kernel developers
- Open source – `https://git-scm.com`
- Distributed
  - No central server required
    - ...but still useful to have one
- Fast
- Data assurance
  - Checksums to make sure you get out what you put in

git vs. GitHub vs. GitLab

- git is an open source software
  - `https://git-scm.com`
- GitHub is a (commercial) web platform
  - Recently bought by Microsoft
  - GitHub provides a central server for git repositories *and* additional services (wiki, ticket system, ...)
  - `https://github.com`
- GitLab is an open source software
  - Provides a central server that you can install on your own server (e.g., at the IMS)
  - `https://about.gitlab.com`

Reading



Scott Chacon and Ben Straub: "Pro Git". 2nd edition. Apress, 2014.
`https://git-scm.com/book/en/v2`

Table of Contents

1. Getting Started
2. Git Basics
3. Git Branching
4. Git on the Server
5. Distributed Git
6. ...

Subsection 1

How does git work?

Commit

- One version of an entire directory (including subdirectories)
- Creating commits is the central activity we do
- Each commit knows its predecessor
- Each commit is identified by a hash value:
  `0eabb4bfef80be2af18255dc19301b989da1f1a3`
- A commit can include changes in multiple files

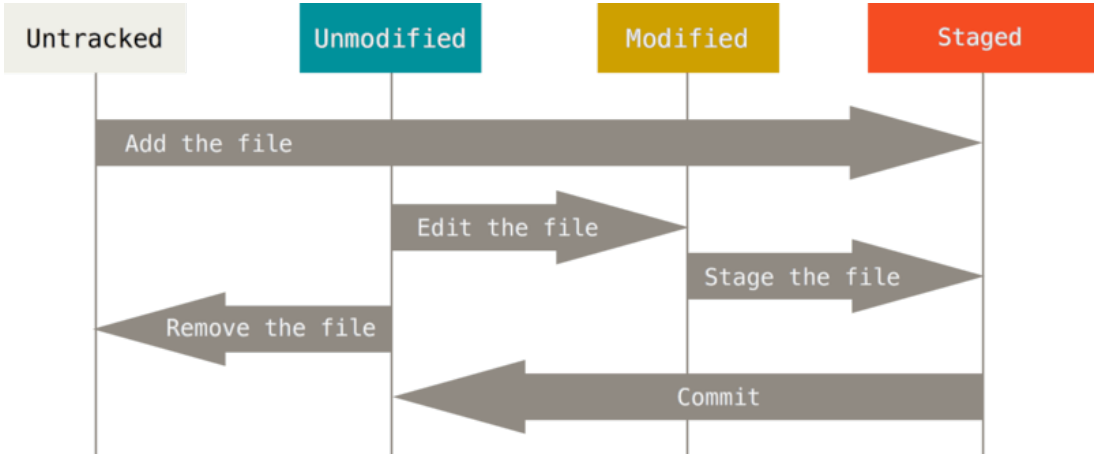


Figure: The lifecycle of the status of your files (Chacon/Straub: Pro Git)

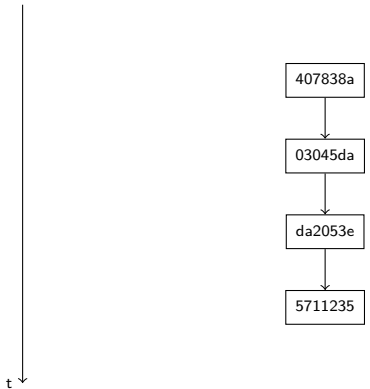
Workflow

1. (Pull changes from others)
2. Edit/add files
3. Put files in staging area
   - `git add <FILENAME>`
   - `git remove <FILENAME>`
4. Commit all files in staging area
   - Provide a useful description
   - `git commit -m "comment"`
5. (Push to others)

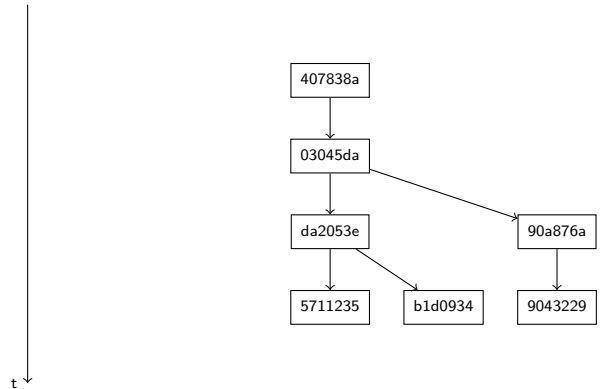
Branching

- Maintaining multiple branches is often useful
- At each time, a single branch is active
  - By default: `master` or `main`
- Switch to an existing branch
  - `git checkout <BRANCHNAME>`
  - To create a new branch, add the option `-b`:
    - `git checkout -b <BRANCHNAME>`

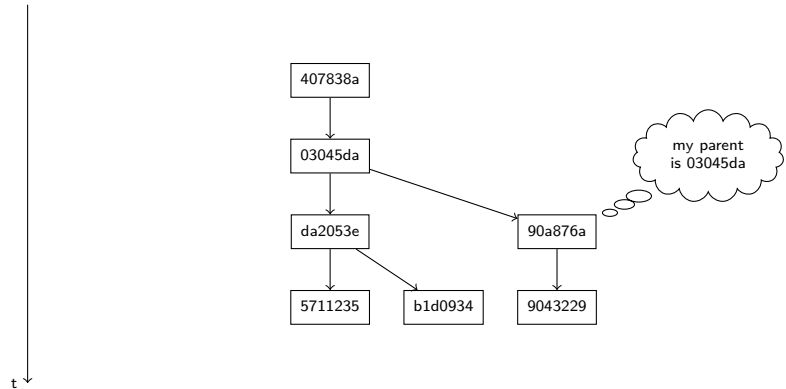
Branching and committing results in a tree



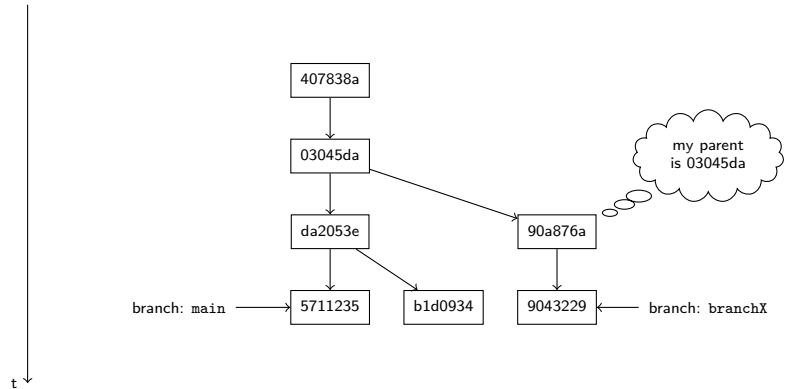
Branching and committing results in a tree



Branching and committing results in a tree



Branching and committing results in a tree



demo

Repository vs. working copy

- The git repository keeps track of *all* past versions and branches
- The working copy can be set to any of the past versions
- `git checkout REFNAME`
  - `REFNAME` can be a branch or revision hash (or tag)
- Checking out moves the HEAD pointer to another revision
  - The HEAD pointer always points to the revision that's active in your working copy

Remotes

- Git repositories can be associated with *remote* repositories
  - Remote repositories are usually on a different computer (e.g., GitHub)

Remotes

- Git repositories can be associated with *remote* repositories
  - Remote repositories are usually on a different computer (e.g., GitHub)
- A repository needs to be synchronized with its remote manually:
  - `git push`: Transfers the commits on the local branch to the same branch on the remote
  - `git pull`: Transfers the commits on the remote branch to the local branch
  - `git clone REPOURL`: Create a local copy of the repository url, setting REPOURL as 'origin' remote

Useful commands

`git status`

Shows the status of the current working copy

- Changed files
- Files in the staging area
- The current branch

Useful commands

`git status`

Shows the status of the current working copy

- Changed files
- Files in the staging area
- The current branch

`git log`

Shows information about current and past commits
Useful options:

`--oneline` Each commit is shown on a single line

`--graph` Information is rendered visually

`--all` Shows information about all branches

On GUIs

Git has a complex task and is a complex piece of software

- Graphical user interfaces do exist and make some tasks easier
- In this class: command line

On GUIs

Git has a complex task and is a complex piece of software

- Graphical user interfaces do exist and make some tasks easier
- In this class: command line
- Recommendations
  - SourceTree (Win/Mac): `https://www.sourcetreeapp.com`
    - Needs a registration with BitBucket (similar to GitHub), but free
  - GitKraken (Win/Mac/Lin): `https://www.gitkraken.com`
    - Free for open source projects

On GUIs

Git has a complex task and is a complex piece of software

- Graphical user interfaces do exist and make some tasks easier
- In this class: command line
- Recommendations
  - SourceTree (Win/Mac): `https://www.sourcetreeapp.com`
    - Needs a registration with BitBucket (similar to GitHub), but free
  - GitKraken (Win/Mac/Lin): `https://www.gitkraken.com`
    - Free for open source projects
- More can be found here:
  `https://git-scm.com/downloads/guis/`

Exercise 01

https://github.com/idh-cologne-deep-learning/exercise-01