Toolbox Workshop 2013

Igor Babuschkin Kevin Dungs Ismo Toijala

2. Oktober 2013

technische universität
dortmund

PEP ET AL. E.V.
PHYSIKSTUDIERENDE UND
EHEMALIGE PHYSIKSTUDIERENDE
DER TU DORTMUND

PEP ET AL. E.V.

PHYSIKSTUDIERENDE UND
EHEMALIGE PHYSIKSTUDIERENDE
DER TU DORTMUND

www.pep-dortmund.org

- Absolventen, Studierende, Mitarbeiter, Freunde und Förderer der Fakultät Physik
- Mission: Netzwerk aufbauen

PeP et al. – Aktivitäten

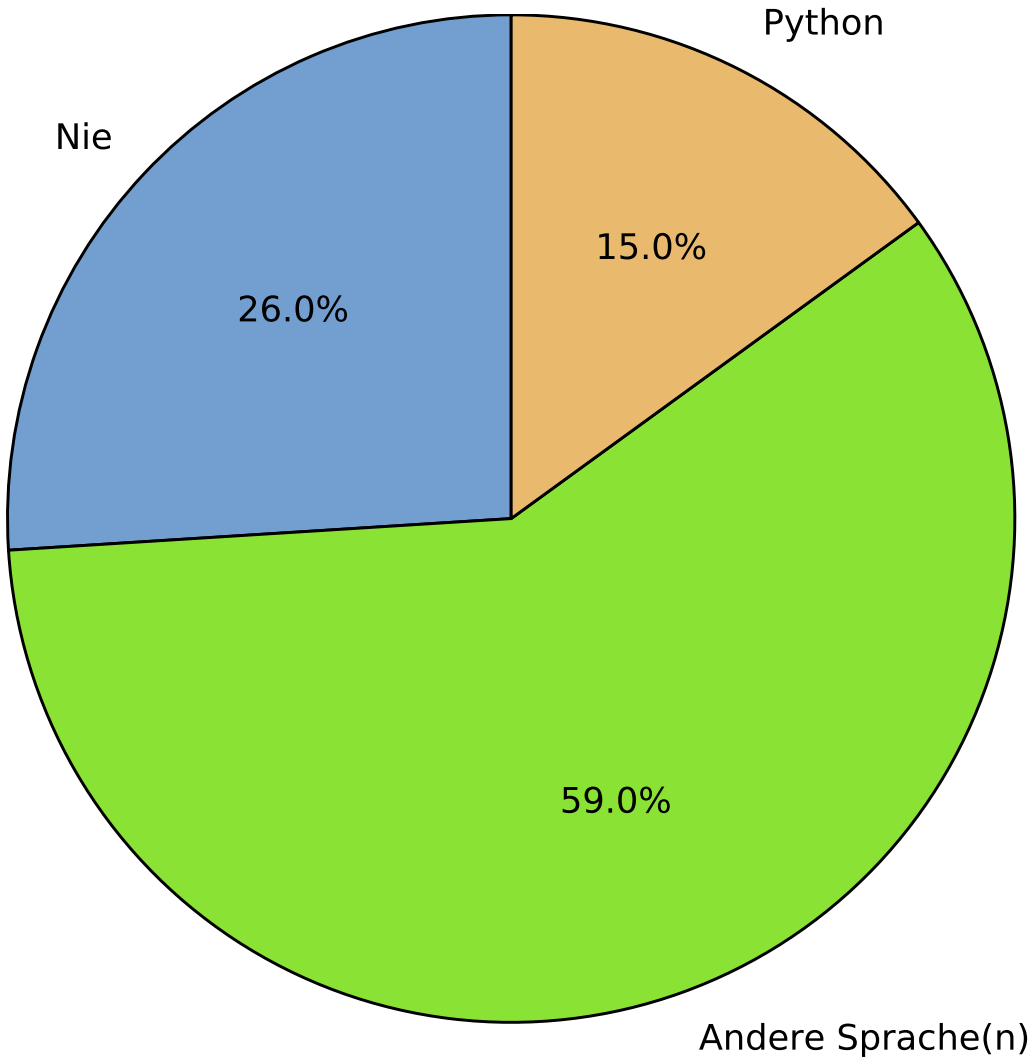
- Stipendien
- Stammtisch
- Bachelorkolloquium
- Absolventenfeier
- Rhetorik-Workshop
- Toolbox-Workshop
- Sommerakademie
- ...

Motivation

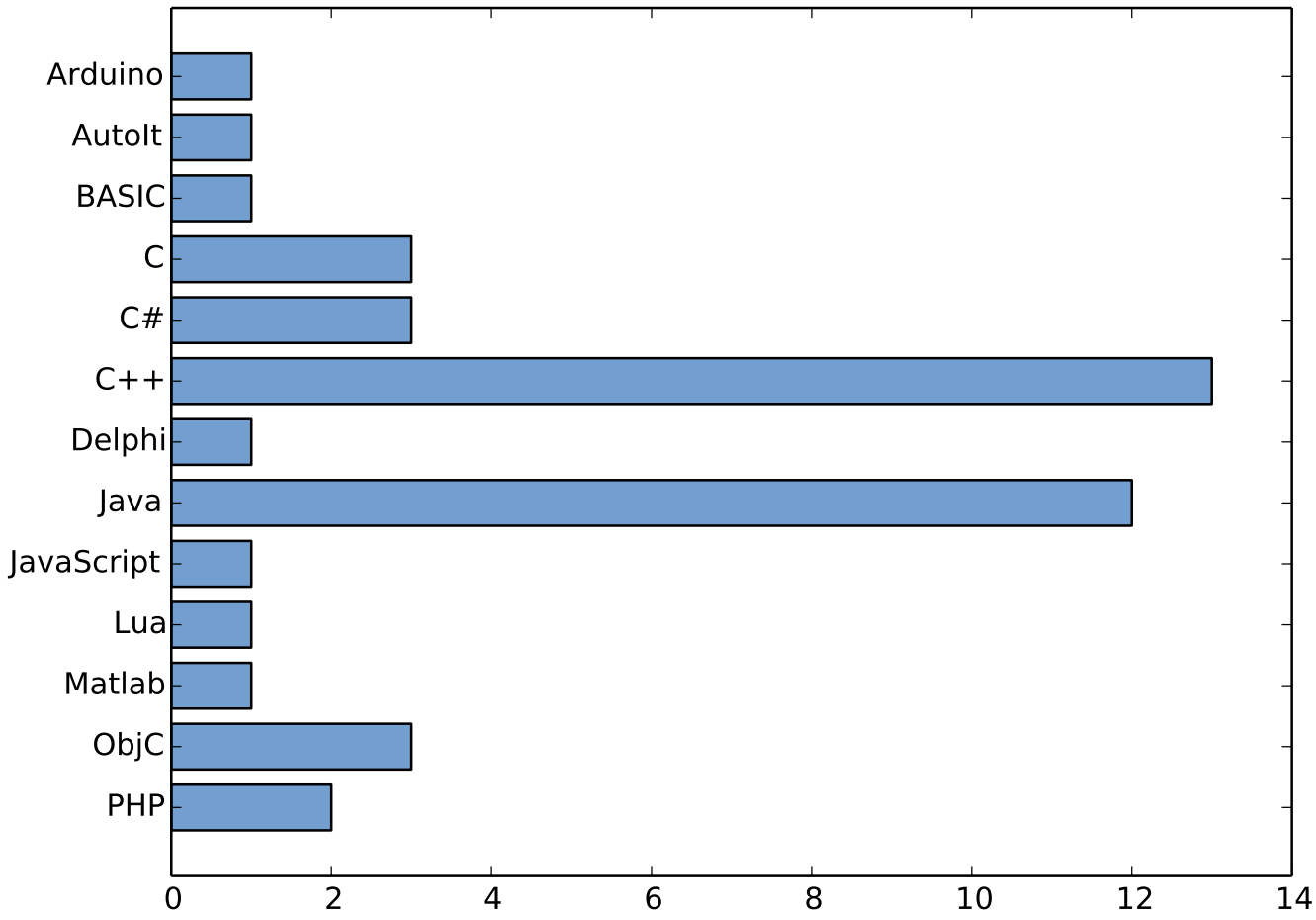
- Arbeitserleichterung
- “Gute” Tools
  - Kein Excel!
- Teamarbeit++
- *best practices*

Auswertung der Umfrage

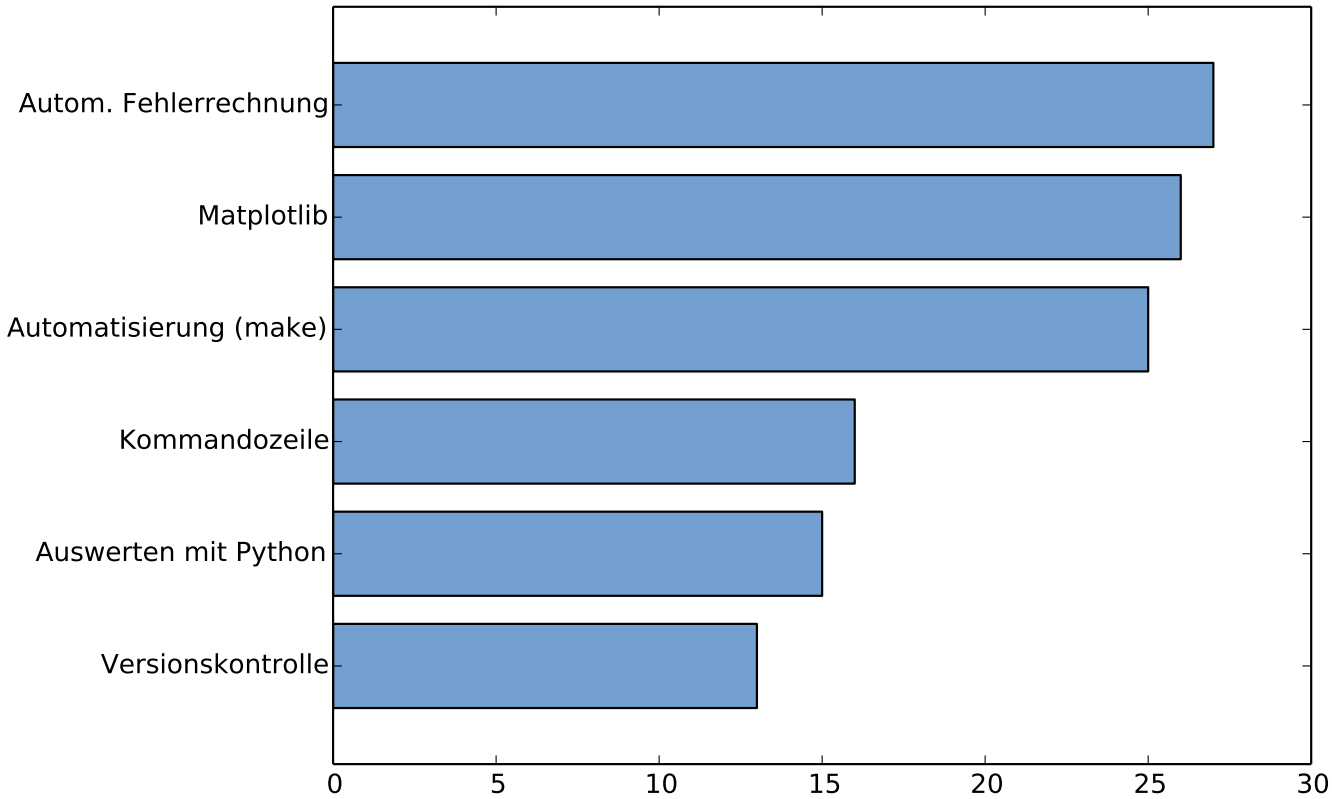
Vorherige Programmierkentnisse?



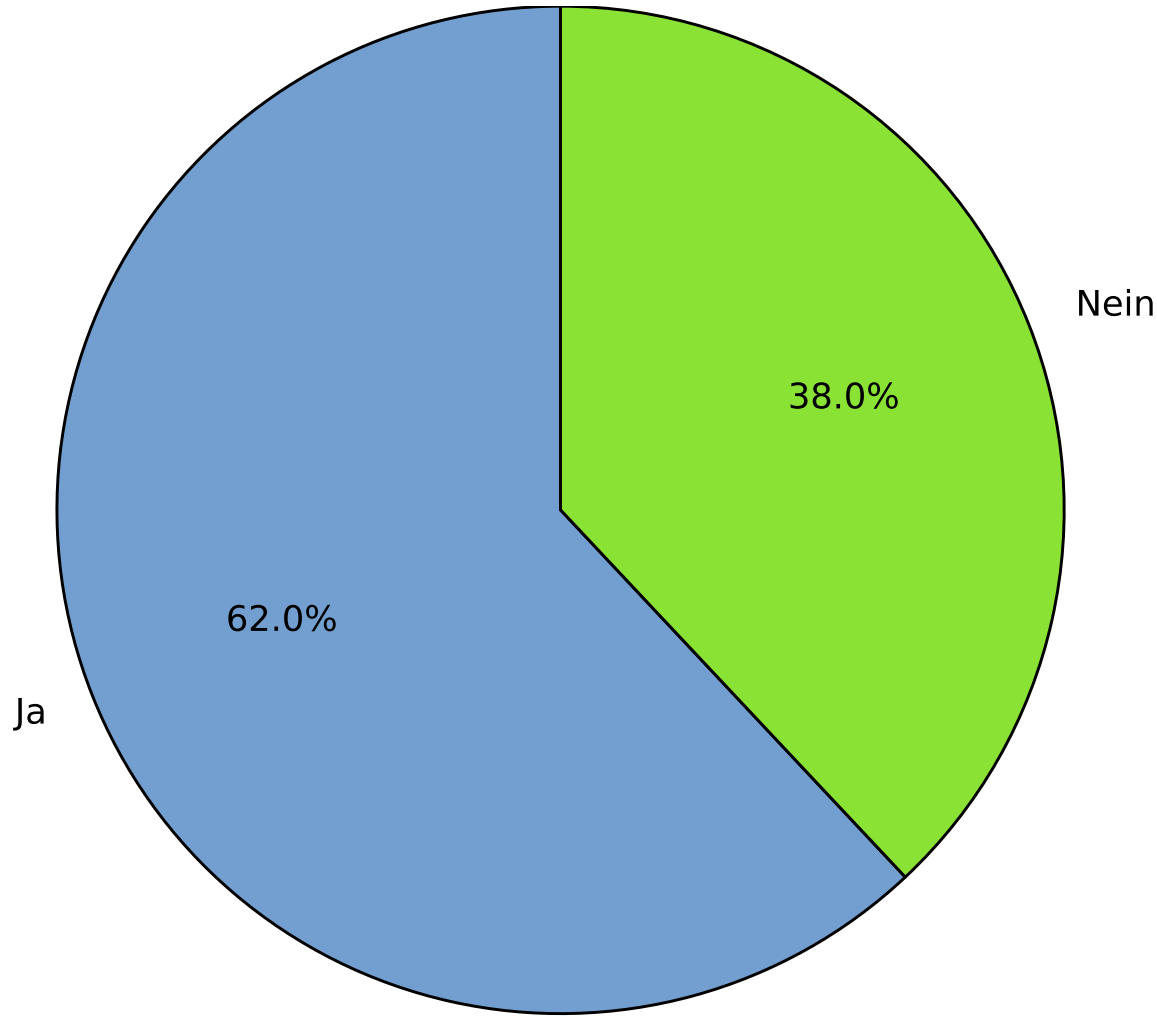
Bekannte Programmiersprachen



Besondere Interessen



Virtuelle Maschine?

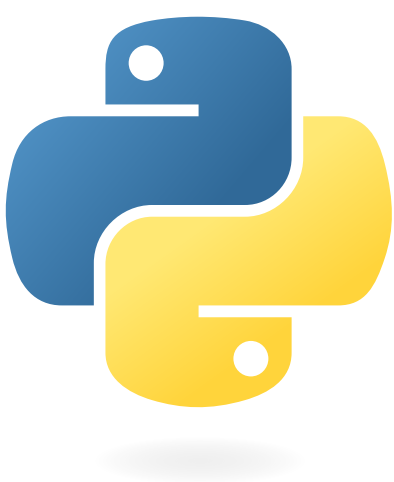


Was mich sonst noch interessiert

- Alles, was ich fürs 3. Semester brauche
- Warum matplotlib, wenn wir eine Woche später GnuPlot kennenlernen?
- Braucht man das wirklich alles?
- Umgang mit Linux
- Warum soll Ubuntu/Linux soviel besser als Windows sein?

Wissenschaftliches Arbeiten mit Python

Warum Python?



python™

- Leicht zu lernen
- Sehr produktiv (man ist schneller fertig)
- Häufig in der Wissenschaft verwendet (gute Bibliotheken)
- Guter interaktiver Interpreter (`IPython`)

Themen

1. Plots Erstellen mit Python (`matplotlib`)
2. Einführung in die Programmierung mit Python
3. Wissenschaftliche Bibliotheken: Numpy, SciPy

Zwischendurch: kurze Aufgaben

Aber zuerst: Umgang mit

`IPython`

Unix Shell

Dateisystem

- bildet *einen* Baum
  - beginnt bei / (root)
  - / trennt Teile eines Pfads
  - auf Groß-/Kleinschreibung achten!
- es gibt ein aktuelles Verzeichnis
- relative vs. absolute Pfade
- spezielle Verzeichnisse:
  - . das aktuelle Verzeichnis
  - .. das Oberverzeichnis
  - ~ das Homeverzeichnis

`man, pwd, cd`

`man` *`topic`*	„manual“: zeigt die Hilfe für ein Programm
`pwd`	„print working directory“: zeigt das aktuelle Verzeichnis
`cd` *`directory`*	„change directory“: wechselt in das angegebene Verzeichnis

`ls`

`ls [`*`directory`*`]`	„list“: zeigt den Inhalt eines Verzeichnisses an
`ls -l`	„long“: zeigt mehr Informationen über Dateien und Verzeichnisse
`ls -a`	„all“: zeigt auch versteckte Dateien (fangen mit `.` an)

mkdir, touch

`mkdir` *directory*	„make directory“: erstellt ein neues Verzeichnis
`mkdir -p` *directory*	„parent“: erstellt auch alle notwendigen Oberverzeichnisse
`touch` *file*	erstellt eine leere Datei

cp, mv, rm, rmdir

`cp` *source destination*	„copy“: kopiert eine Datei
`cp -r` *source destination*	„recursive“: kopiert ein Verzeichnis rekursiv
`mv` *source desination*	„move“: verschiebt eine Datei (Umbenennung)
`rm` *file*	„remove“: löscht eine Datei (Es gibt keinen Papierkorb!)
`rm -r` *directory*	„recursive“: löscht ein Verzeichnis rekursiv
`rmdir` *directory*	„remove directory“: löscht ein *leeres* Verzeichnis

`cat, less, grep, echo`

`cat` *`file`*	„concatenate“: gibt Inhalt einer (oder mehr) Datei(en) aus
`less file`	(besser als `more`): wie `cat`, aber navigabel
`grep` *`pattern file`*	`g/re/p`: sucht in einer Datei nach einem Muster
`grep -i` *`pattern file`*	„case insensitive“
`grep -r` *`pattern directory`*	„recursive“: suche rekursiv in allen Dateien
`echo` *`message`*	gibt einen Text aus

Ein- und Ausgabe

`command > file`	überschreibt Datei mit Ausgabe
`command >> file`	fügt Ausgabe einer Datei hinzu
`command < file`	Datei als Eingabe
`command1 | command2`	Ausgabe als Eingabe (Pipe)

Tastaturkürzel

`Ctrl-C`	beendet das laufende Programm
`Ctrl-D`	EOF (end of file) eingeben, kann Programme beenden
`Ctrl-L`	leert den Bildschirm

Globbing

`*`	wird ersetzt durch alle passenden Dateien
`{a,b}`	bildet alle Kombinationen

Beispiele:

```
*.log            →  foo.log bar.log
foo.{tex,pdf}    →  foo.tex foo.pdf
```

Auswertungen automatisieren mit `make`

30 Jahre GNU!

Automatisierte, reproduzierbare Prozesse

Problem:

- kurz vor Abgabe noch neue Korrekturen einpflegen
  - Tippfehler korrigieren, Plots bearbeiten
- TEX ausführen, ausdrucken
- vergessen, Plots neu zu erstellen

Lösung: Make!

- sieht, welche Dateien geändert wurden
- berechnet nötige Operationen
- führt Python-Skript aus, führt TEX aus

Makefile

- Datei heißt `Makefile`, keine Endung!
  - bei Windows Dateiendungen einschalten! (`http://support.microsoft.com/kb/865219/de`)
- besteht aus Rules:

Rule

```
target: prerequisites
	recipe
```

`target` Datei(en), die von dieser Rule erzeugt wird

`prerequisites` Dateien, von denen diese Rule abhängt

`recipe` Befehle: `prerequisites` → `target` (mit Tab eingerückt)

Beispiel

```
all: report.pdf

plot.pdf: plot.py daten.txt
	python2 plot.py

report.pdf: report.tex plot.pdf
	latex report.tex
```

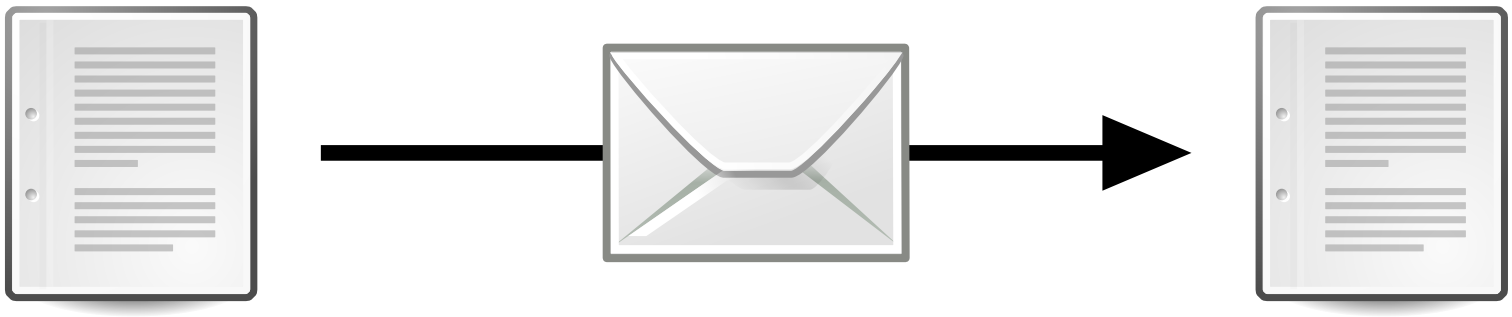
- wenn nur `make` gestartet wird, wird der erste Target erstellt, hier `all`; um ein anderes Target zu erstellen, startet man `make` *`target`*
- `all` braucht `report.pdf`, also wird `latex report.tex` ausgeführt
- `report.pdf` braucht noch `plot.pdf`, also wird Python ausgeführt

Änderungen verwalten mit `git`

Wie arbeitet man am besten an einem Protokoll zusammen?

Idee: Austausch über Mails

Mails: Probleme



- Risiko, dass Änderungen vergessen werden, ist groß
- Bei jedem Abgleich muss jemand anders aktiv werden
  - Stört
  - Es kommt zu Verzögerungen

Fazit: Eine sehr unbequeme / riskante Lösung

Idee: Austausch über Dropbox

Dropbox: Probleme

- Man merkt nichts von Änderungen der Anderen
- Gleichzeitige Änderungen gehen verloren

Fazit: Besser, aber hat deutliche Probleme

Lösung: Änderungen verwalten mit `git`

- Ein Version Control System
- Ursprünglich entwickelt, um den Programmcode des Linux-Kernels zu verwalten (Linus Torvalds)
- Hat sich gegenüber ähnlichen Programmen (SVN, mercurial) durchgesetzt

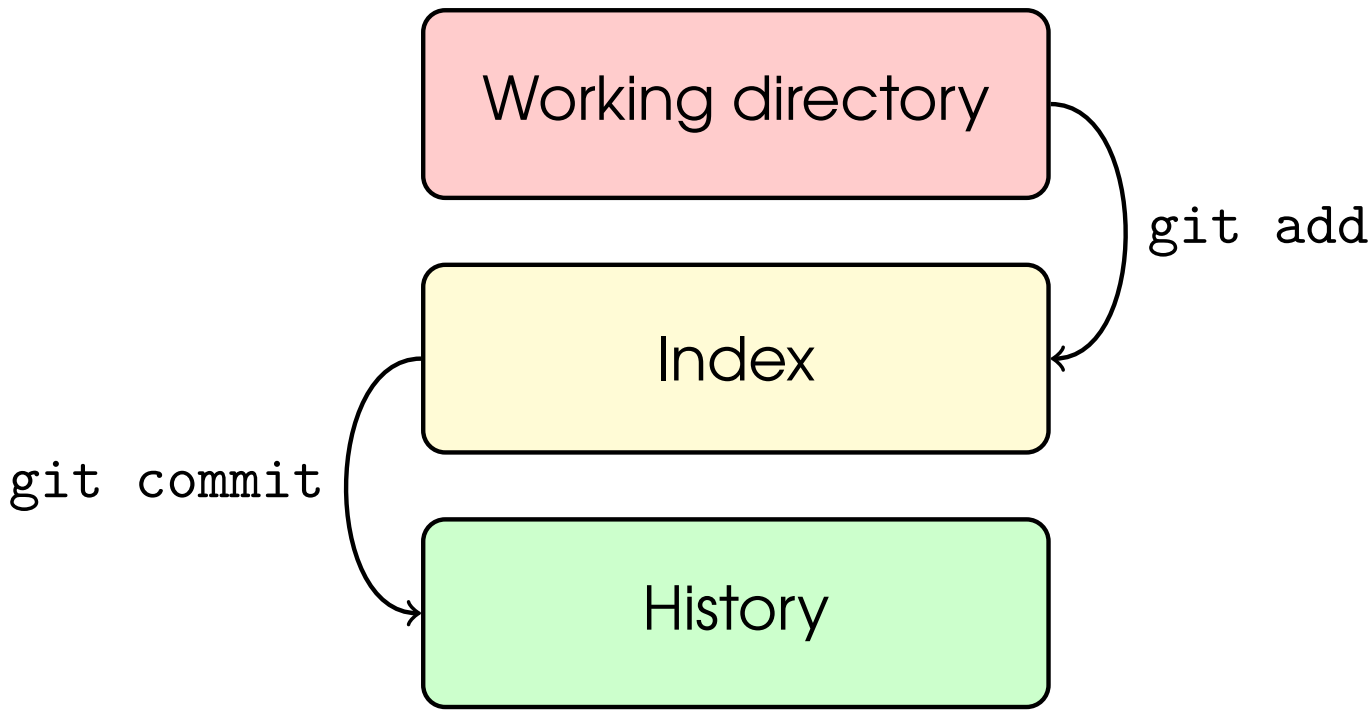
Was bringt `git` für Vorteile?

- Arbeit wird für andere sichtbar protokolliert
- Erlaubt Zurückspringen an einen früheren Zeitpunkt
- Kann die meisten Änderungen automatisch zusammenfügen
- Wirkt nebenbei auch als Backup

Einziges Problem: Man muss lernen, damit umzugehen

Zentrales Konzept: Das Repository

- Erzeugen mit `git init`



Mit anderen Repositories kommunizieren

- Repository kopieren: `git clone`
- Neue Änderungen holen: `git pull`
- Eigene Änderungen hochladen: `git push`

Achtung: Merge conflicts

Don't Panic

- Entstehen, wenn `git` nicht automatisch mergen kann (selbe Zeile geändert, etc.)

1. Die betroffenen Dateien öffnen
2. Markierungen finden und die Stelle selbst mergen (meist 2,3 Zeilen)
3. `git commit` ausführen um zu bestätigen

Weitere nützliche Befehle

- Änderungen ansehen: `git diff`
- Vergangenheit betrachten: `git log`
- Änderungen kurz zur Seite schieben: `git stash`