
PeP et al.
Toolbox Workshop

TU Dortmund
Version 0.9 – 28. September 2012

Igor Babuschkin¹ • Kevin Dungs² • Christian Gerhorst³
Peter Lorenz⁴ • Ismo Toijala⁵

¹igor.babuschkin@udo.edu

²kevin.dungs@udo.edu

³christiangerhorst@gmail.com

⁴peter.lorenz@udo.edu

⁵ismo.toijala@udo.edu

Inhaltsverzeichnis

1	Einleitung	4
1.1	Motivation	4
1.2	PeP et al.	4
1.3	Installation	4
1.3.1	Windows 7	5
1.3.2	Mac OS X	6
1.3.3	Ubuntu 12.04	6
1.3.4	Arch Linux	7
1.3.5	Einstellungen und Testen	7
2	Unix Shell	8
2.1	Dateisystem	8
2.2	Aufbau einer Eingabe	8
2.3	Befehle	9
2.3.1	man	9
2.3.2	pwd, cd	9
2.3.3	ls	9
2.3.4	mkdir, touch	10
2.3.5	cp, mv, rm, rmdir	11
2.3.6	cat, less, grep, echo	11
2.4	Nützliche Shell Features	12
2.4.1	Tastaturkürzel	12
2.4.2	Ein- und Ausgabe	12
2.4.3	Globbering (*)	13
2.5	Weitere nützliche Befehle	13
2.6	Weiterführende Links	13
3	Git	14
3.1	Warum Versionskontrolle?	14
3.2	Überblick	15
3.3	Hosting	15
3.4	Befehle	15
3.4.1	git init	15
3.4.2	git clone	16
3.4.3	git status	16
3.4.4	git add	16

3.4.5	git log	16
3.4.6	git rm, git mv	16
3.4.7	git commit	16
3.4.8	git push	17
3.4.9	git pull	17
3.4.10	git mergetool	17
3.4.11	.gitignore	17
3.5	Best Practices	17
3.6	Weiterführende Links	18
4	Python	19
4.1	IPython	19
4.2	Syntax	20
4.2.1	Variablen	20
4.2.2	Datenstrukturen/-typen	20
4.2.3	Operatoren	21
4.2.4	Dateien	22
4.3	Bibliotheken	22
4.3.1	NumPy	22
4.3.2	SciPy	23
4.3.3	matplotlib	23
4.4	Python 2	24
4.5	Weiterführende Links	24
5	L^AT_EX	26
5.1	useful packages	26
5.1.1	must	26
5.1.2	should	26
5.1.3	can	27

1 Einleitung

1.1 Motivation

<http://www.americanscientist.org/issues/pub/wheres-the-real-bottleneck-in-scientific-computing>

1.2 PeP et al.



Der Verein versteht sich als Einrichtung für Absolventen, Studierende, Mitarbeiter sowie für Freunde und Förderer der Fakultät Physik der TU Dortmund. Gegründet auf Initiative einiger Absolventen ist es seine Aufgabe, ein Netzwerk zwischen den Absolventen und der Fakultät aufzubauen.

- Networking
- Stammtisch
- Rhetorik-Seminar
- Stipendien
- Sommerakademie
- ...

<http://pep-dortmund.org>

1.3 Installation

Ein \ am Ende einer Eingabezeile bedeutet, dass die Zeile aus Platzgründen gebrochen wurde. Man sollte das \ weglassen und die Zeile ganz, ohne Umbruch eingeben. Der Abschnitt 1.3.5 gilt für alle Unix-artigen Betriebssysteme, also sowohl Linux als auch Mac OS X.

1.3.1 Windows 7

Downloads

Ein Archiv mit den benötigten Installationsdateien kann unter <http://files.babushk.in/toolbox.zip> heruntergeladen werden.

Zur Installation einfach die Installationsdateien nacheinander starten.

Git

- Git-1.7.11-preview20120710.exe
- Pfad: C:\Programs\Git
- Use a TrueType font in all console windows

KDiff3

- KDiff3-32bit-Setup_0.9.97.exe
- Pfad: C:\Programs\KDiff3

Python

- python-3.2.3.msi
- Pfad: C:\Programs\Python32

Numpy, Scipy, Matplotlib, Distribute, Pyreadline, Pyzmq, PyQt4, Pygments

- numpy-1.6.2-win32-superpack-python3.2.exe
- scipy-0.10.1-win32-superpack-python3.2.exe
- matplotlib-1.2.0rc2.win32-py3.2.exe
- distribute-0.6.28.win32-py3.2.exe
- pyreadline-2.0-dev1.win32-py3.2.exe
- pyzmq-2.2.0.win32-py3.2.msi
- PyQt-Py3.2-x86-gpl-4.9.4-1.exe
- Pygments-1.5.win32-py3.2.exe

IPython

- ipython-0.13.py3-win32.exe
- Rechtsklick → Als Administrator ausführen

Letzte Schritte

- Rechtsklick auf `install.bat` → Als Administrator ausführen
- Git Bash starten und folgende Befehle eintippen (und dabei Name und Email eintragen):

```
git config --global user.email "XXX@udo.edu"  
git config --global user.name "XXX"
```

Test

Jetzt kann man noch testen, ob alles vernünftig funktioniert:

- Start → Programme → IPython (Py3.2 32 bit) → pylab
- `plot([1, 2, 3])`

Nach dieser Eingabe sollte ein Plot erscheinen.

1.3.2 Mac OS X

Am einfachsten ist die Installation bei OS X unter Verwendung von Mac Ports. Auf der Website <http://www.macports.org/> gibt es ein entsprechendes .dmg. Dabei ist zu beachten, dass XCode installiert sein muss. Sollte kein XCode vorhanden sein, ist es auf der OS X Installations-DVD zu finden oder kann über den AppStore heruntergeladen werden (OS X 10.7 vorausgesetzt).

Ist Mac Ports erst einmal installiert, ist die Installation der benötigten Programme und Bibliotheken denkbar einfach. In der Kommandozeile (z.B. Terminal.app) wird einfach

```
sudo port install gmake git-core python27 py27-ipython py27-numpy \  
py27-scipy py27-matplotlib kdifff3 py27-pyqt4 py27-sip py27-pygments
```

einggegeben. Das Ausführen von

```
sudo port select python python27
```

sorgt dafür, dass die neu installierte Python-Version als Standard verwendet wird und nicht die von Apple mitgelieferte.

Mitte Oktober 2012 wird eine neue Version von Matplotlib erscheinen. Diese wird auch mit Python 3 funktionieren und über MacPorts erhältlich sein. Sobald diese Versionen verfügbar sind, wird ihre Verwendung empfohlen. Diese Anleitung wird dann entsprechend aktualisiert.

1.3.3 Ubuntu 12.04

Die benötigten Programme und Libraries kann man unter Ubuntu am schnellsten per Kommandozeile installieren.

Git

Die Versionskontrolle 'Git' installiert man mit dem Befehl

```
sudo apt-get install git kdiff3
```

Python und Bibliotheken

Zusätzlich zu Python 3 sollte man die Python-Bibliotheken NumPy und SciPy (für wissenschaftliche Berechnungen) installieren. Bei IPython handelt es sich um eine interaktive Konsole für Python, mit der man, ähnlich wie mit der Kommandozeile, Befehle oder Skripte ausführen kann. Der Installationsbefehl lautet

```
sudo apt-get install python3 python3-numpy python3-scipy ipython3 \  
python3-pyqt4 python3-zmq python3-pygments python3-sip
```

Matplotlib

Leider gibt es noch kein Paket für die Python3-Version von Matplotlib (zum Erstellen von Plots). Es gibt aber ein externes Repository mit der neuesten Version:

```
sudo add-apt-repository ppa:takluyver/matplotlib-daily  
sudo apt-get update  
sudo apt-get install python3-matplotlib
```

1.3.4 Arch Linux

```
sudo pacman -S git python python-numpy python-scipy \  
ipython pyqt sip python-pygments python-pyzmq kdiff3  
yaourt -S python-matplotlib-git
```

1.3.5 Einstellungen und Testen

Git und Matplotlib einstellen (eigene Daten eintragen):

```
git config --global user.email "XXX@udo.edu"  
git config --global user.name "XXX"  
mkdir -p ~/.matplotlib  
echo 'backend : Qt4Agg' > ~/.matplotlib/matplotlibrc  
echo "  
[merge]  
    tool = kdiff3" >> ~/.gitconfig
```

Jetzt kann man noch testen, ob alles vernünftig funktioniert:

```
ipython3 --pylab  
plot([1,2,3])
```

Nach dieser Eingabe sollte ein Plot erscheinen.

2 Unix Shell



Unix Shell

2.1 Dateisystem

Das Dateisystem in Linux besteht aus *einem* Baum. Dies äußert sich im so genannten Dateipfad, der den Weg von der Wurzel / zur eigentlichen Datei oder dem Ordner beschreibt. Nach jedem Knoten, das heißt nach jedem Überordner setzt man wieder ein Slash bis man beim Dateinamen angekommen ist.

Es gibt immer ein aktuelles Verzeichnis, in dem man sich befindet; dieses wird 'working directory' genannt. Will man aus dem working directory in ein anderes Verzeichnis so kann man dies durch eine relative oder absolute Pfadangabe tun. Bei einem absoluten Pfad geht man wieder von der Wurzel aus und beginnt ihn mit /. Wählt man den relativen Pfad, der oft kürzer ist, insbesondere wenn man einfach nur in den Über- oder Unterordner möchte, so baut man die Pfadangabe ausgehend vom aktuellen Verzeichnis auf. Bei der Pfadangabe sollte man im Übrigen folgende besondere Verzeichnisse kennen:

- . das aktuelle Verzeichnis (oder der aktuelle im bisherigen Pfad, $a/. / = a/$)
- .. das Oberverzeichnis des aktuellen Verzeichnisses ($a/b/.. / = a/$)
- ~ das Heimverzeichnis (nur am Anfang eines Pfads)

2.2 Aufbau einer Eingabe

```
$ ls -l --all directory  
output  
$
```

\$	Prompt
ls	Befehl
-l	kurze Option
--all	lange Option
<i>directory</i>	Argument
<i>output</i>	Ausgabe

- \$ ist nur ein Beispiel für einen Prompt, häufig wird das aktuelle Verzeichnis und/oder andere Informationen angezeigt
- kurze Optionen können zusammengefasst werden (`ls -la = ls -l -a = ls -l --all`)
- die Reihenfolge der Optionen ist egal
- meistens werden mehrere Argumente (z.B. Dateien) akzeptiert

2.3 Befehle

2.3.1 man

- `man topic` für manual: zeigt die Hilfe für ein Programm
- Beispiel: `man man`

2.3.2 pwd, cd

- `pwd` für print working directory: zeigt das aktuelle Verzeichnis
- `cd directory` für change directory: wechselt in das angegebene Verzeichnis
- Beispiel:

```
$ pwd
/home/ismo
$ cd ../../etc
$ pwd
/etc
$ cd ~
$ pwd
/home/ismo
```

2.3.3 ls

- `ls [directory]` für list: zeigt den Inhalt eines Verzeichnisses an
- `ls -l`: zeigt mehr Informationen über Dateien und Verzeichnisse

- `ls -a`: zeigt auch versteckte Dateien
- `ls -R`: zeigt auch den Inhalt von Unterverzeichnissen
- alle Optionen können kombiniert werden

Beispiel:

```
$ ls
a/ b
$ ls -l
total 4.0K
drwxr-xr-x 2 ismo users 4.0K Sep 15 19:52 a/
-rw-r--r-- 1 ismo users  0 Sep 15 19:52 b
$ ls -a
./ ../ a/ b
$ ls -R
.:
a/ b

./a:
c
```

2.3.4 mkdir, touch

- `mkdir directory` für make directory: erstellt ein neues Verzeichnis
- `mkdir -p directory`: erstellt ein neues Verzeichnis und alle notwendigen Oberverzeichnisse
- `touch file`: erstellt eine neue, leere Datei

Beispiel:

```
$ ls
$ mkdir a
$ mkdir b/c
mkdir: cannot create directory 'b/c': No such file or directory
$ mkdir -p b/c
$ touch b/file
$ ls -R
.:
a/ b/
~
./a:
~\
./b:
```

c/ file

./b/c:

2.3.5 cp, mv, rm, rmdir

- `cp source destination` für copy: kopiert eine Datei
- `cp -r source destination`: kopiert ein Verzeichnis rekursiv
- das Ziel kann ein Verzeichnis oder der exakte Pfad sein
- `source` können mehrere Dateien sein, der letzte Pfad zählt als *destination*
- `mv source destination` für move: verschiebt oder benennt eine Datei um
- Ziel kann wie bei `cp` sein
- `rm file` für remove: löscht eine Datei
- `rm -r file`: löscht ein Verzeichnis rekursiv
- `rmdir directory` für remove directory: löscht ein leeres Verzeichnis
- `rm -r` kann statt `rmdir` verwendet werden

Beispiel:

```
$ ls
a
$ cp a b
$ ls
a b
$ mv b c
$ ls
a c
$ rm a
$ ls
c
```

2.3.6 cat, less, grep, echo

- `cat [file]` für concatenate: gibt den Inhalt einer (oder mehrerer) Dateien aus
- `less [file]` (besser als `more`): zeigt eine Datei in einer navigablen Form an
- `grep pattern [file]` für ???: sucht nach einem Muster
- `grep -i pattern [file]`: ignoriert Groß- und Kleinschreibung

- `grep -r pattern directory`: sucht rekursiv in allen Dateien
- `echo text`: gibt einen Text aus

2.4 Nützliche Shell Features

2.4.1 Tastaturkürzel

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

2.4.2 Ein- und Ausgabe

- Programme haben einen Eingabe- und einen Ausgabestream
- diese können verschieden verwendet werden
- `command > file`: überschreibt eine Datei mit der Ausgabe
- `command >> file`: wie `>`, aber schreibt ans Ende der Datei, statt zu überschreiben
- `command < file`: verwendet eine Datei als Eingabe
- `command1 | command2`: verwendet die Ausgabe eines Programms als Eingabe eines anderen

Beispiel:

```
$ echo "bla bla bla" > a
$ cat a
bla bla bla
$ echo "foo" >> a
$ cat a
bla bla bla
foo
$ grep foo < a
foo
$ touch abc
$ touch bcd
$ touch cde
$ ls | grep b
abc
bcd
```

2.4.3 Globbing (*)

- * wird ersetzt durch alle passenden Dateien
- Beispiel:

```
$ ls .vim*  
.viminfo .vimrc  
  
.vim:  
backup/ bundle/ swap/ syntax/
```

2.5 Weitere nützliche Befehle

- **vim**: Ein sehr mächtiger Texteditor in der Kommandozeile
- **vimtutor**: Einführung in vim
- **emacs**: Noch ein Editor
- **nano**: Einfacher Editor in der Kommandozeile
- **find, locate**: Finden Dateien und Verzeichnisse
- **curl, wget**: Dateien herunterladen
- **rsync**: Synchronisierung von Verzeichnisstrukturen
- **awk, sed**: Bearbeitung von Text
- **ln**: Erstellt Links zwischen Dateien und Verzeichnissen
- **diff**: Zeigt unterschiede zwischen Dateien
- **patch**: Wendet einen von diff erstellten Diff an

2.6 Weiterführende Links

- Beginning with the Shell: <http://youtu.be/Sye3mu-EoTI>
- Learn CLI The Hard Way: <http://cli.learncodethehardway.org/book/>
- Bash Guide for Beginners: <http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html>
- Advanced Bash-Scripting Guide: <http://tldp.org/LDP/abs/html/index.html>
- Zsh Workshop: <http://www.acm.uiuc.edu/workshops/zsh/toc.html>
- Zsh Manual: http://zsh.sourceforge.net/Doc/Release/zsh_toc.html

3 Git



3.1 Warum Versionskontrolle?

Wissenschaftliche Projekte werden meist in Kollaboration durchgeführt. Die einzelnen Wissenschaftler arbeiten dabei oft an völlig verschiedenen Orten. Es müssen längere Dokumente verfasst werden; oft werden Programme entwickelt. Arbeit und Ergebnisse sollten klar protokolliert und reproduzierbar sein. Auch sollten stets Backups des Projekts existieren.

Ein Versionskontroll-System ist ein Programm, das genau diese Aufgaben erfüllt. Das Konzept der Versionskontrolle kam ursprünglich in der Software-Entwicklung auf. Hier besteht wegen großer Teams und vieler Code-Zeilen ein besonderer Bedarf.

Diese Programme können aber auch außerhalb der Software-Entwicklung effektiv eingesetzt werden.

Bekanntere Versionskontroll-Systeme sind z.B.:

- CVS
- SVN
- Mercurial (`hg`)
- `git`

Besonders `git` setzt sich in den letzten Jahren immer mehr durch. Es weist einige nützliche Vorteile auf:

- Es ist sehr schnell
- Jeder Nutzer besitzt lokal die gesamte Vergangenheit des Projekts. (*Distributed Version Control System*) So kann auch ohne Internetverbindung effektiv gearbeitet werden.
- Das automatische Zusammenfügen mehrerer Änderungen (*mergen*) ist sehr gut gelöst.
- Es existieren gute Online-Services, die `git` unterstützen.

3.2 Überblick

Git ist vorrangig ein Kommandozeilenprogramm. Es ist aber kein Problem wenn man die Kommandozeile lieber meidet: Es existieren zahlreiche grafische Oberflächen, die einem die tägliche Arbeit erleichtern können. Trotzdem sollte man zuerst die Kommandozeilen-Befehle lernen, da man so am schnellsten ein Gefühl für die Funktionsweise von `git` bekommt. Außerdem nutzen Beispiele und Erklärungen im Internet fast ausschließlich die Kommandozeilen-Syntax.

Einige Begriffe sollten man kennen, bevor man sich mit den Befehlen vertraut macht:



Ein `git`-Repository existiert innerhalb eines Verzeichnisses (dem *Working directory*) im Dateisystem. Bis auf den Ordner `.git` innerhalb dieses Verzeichnisses merkt man erst einmal nichts von `git`.

Hat man einige Änderungen an seinem Projekt vorgenommen und möchte diese speichern oder mit anderen teilen, so fügt man sie mit einem Befehl zum sogenannten *Index* hinzu. Ist man mit der Sammlung zufrieden, kann man sie als *Commit* abspeichern. Dabei gibt man in der Regel eine kurze Erklärung, was am Projekt geändert wurde. (Mehrere Commits nacheinander bilden eine Kette)

Man kann seine Commits anschließend hochladen. Interessant wird es, wenn man die Commits anderer herunterlädt. Git fügt diese dann automatisch mit den eigenen zusammen.

3.3 Hosting

Es lohnt sich, ein Hauptrepository im Internet zu haben, auf das alle zugreifen können. Die bekanntesten `git`-Hoster sind

-  **Github** <https://github.com/>
-  **Bitbucket** <https://bitbucket.org/>

Github bietet kostenlose Repositories an, wenn man den Inhalt öffentlich macht. Bitbucket bietet zusätzlich auch kostenlose private Repositories an, die man nur mit einem Passwort einsehen kann. Wer einen eigenen Server besitzt kann `git` auch darauf einrichten. Hier empfiehlt es sich, `gitolite` zu benutzen.

3.4 Befehle

3.4.1 `git init`

Mit diesem Befehl wird aus dem jetzigen Verzeichnis ein leeres Repository.

3.4.2 `git clone`

Kopiert ein Repository (meist aus dem Internet). Als Parameter muss die URL des Repositories übergeben werden. Beispiel:

```
git clone http://github.com/user/repo
```

3.4.3 `git status`

Zeigt an, welche Dateien seit dem letzten Commit verändert wurden, welche neu hinzugekommen sind und welche bereits für den nächsten Commit vorbereitet wurden.

3.4.4 `git add`

Fügt eine Datei oder einen Ordner zum *Index* hinzu. Das bedeutet, dass die Dateien beim nächsten Commit abgespeichert werden. Bei Ordnern werden auch alle Unterordner und Dateien darin hinzugefügt. Wenn man also

```
git add .
```

im obersten Verzeichnis des Repositories aufruft, werden alle Dateien hinzugefügt.

3.4.5 `git log`

Zeigt alle Commits im Repository an. Man kann die Nachricht sehen, mit der sie gespeichert wurden, sowie wann und von wem sie erstellt wurden. Es gibt zahlreiche Optionen, mit denen man die Ausgabe anpassen kann. (siehe `git help log`)

3.4.6 `git rm`, `git mv`

Wenn man Dateien löscht oder sie bewegt muss man git mitteilen, dass man die Datei (an der alten Stelle) tatsächlich löschen will. Das geht am Besten in einem Schritt, wenn man sofort `git rm` und `git mv` benutzt.

3.4.7 `git commit`

Speichert die Änderungen im Index als Commit ab. Öffnet auch einen Editor, damit man beschreiben kann, was man geändert hat. Meistens reicht es aber auch die Änderung mit dem Parameter `-m` direkt einzugeben:

```
git commit -m "Ein paar Änderungen"
```

Das geht schneller.

3.4.8 git push

Schickt die neuen Commits zu dem Repository, das man zu Anfang mit `git clone` kopiert hat. Hat man das Repository ursprünglich mit `git init` erstellt, so muss man erstmal ein Zielrepository einstellen:

```
git remote add --track master origin [URL]
git push origin master
```

Danach ist das Repository abgespeichert und man kann immer `git push` benutzen.

Wenn bereits ein Anderer neue Änderungen ins Repository hochgeladen hat, kommt eine Fehlermeldung. Nun muss man erst einmal `git pull` benutzen (nächster Befehl) und sicherstellen, dass die Änderungen vernünftig zusammengefügt werden. Danach kann man das Ganze mit `git push` hochladen.

3.4.9 git pull

Holt neue Änderungen aus dem Repository. Man kann sich dann mit `git log` ansehen, was gemacht wurde. Hat man im eigenen Repository noch neue Änderungen, so wird `git` versuchen, diese automatisch mit den neuen zu mergen.

3.4.10 git mergetool

Wurde dabei dieselbe Zeile von mehreren Leuten geändert, so kommt es zu einem *merge conflict*. Hierbei sollte man `git mergetool` eingeben, woraufhin sich ein Programm öffnet mit dessen Hilfe man die kritischen Stellen finden und das Problem manuell beseitigen kann. Wer die Installationsanleitung befolgt hat, hat das Programm `kdiff3` installiert, was genau dafür geschrieben wurde. Hinterher muss man noch `git commit` eingeben, um den merge abzuschließen.

3.4.11 .gitignore

Wen es stört, dass Dateien, die man gar nicht im Repository haben will, ständig bei `git status` angezeigt werden, kann die Datei `.gitignore` im obersten Verzeichnis des Repositories erstellen. Darin kann man Regeln definieren, nach denen Dateien von `git` ignoriert werden sollen:

```
*.pdf
build/*
```

In diesem Beispiel werden alle Dateien ignoriert, die auf `.pdf` enden, oder die in einem Ordner mit dem Namen `build` liegen.

3.5 Best Practices

Einige Punkte, die man bei der Benutzung von `git` beachten sollte:

- Automatisch generierte, sich oft ändernde Dateien sollten möglichst nicht im Repository gespeichert werden \Rightarrow bläht sonst die Größe auf
- Git funktioniert am Besten mit reinen Text-Dateien (L^AT_EX-Code, Programme, etc.). Binäre Formate wie Word- oder Excel-Dateien können zwar auch abgespeichert werden, das automatische Mergen funktioniert aber kaum, die Änderungen können nicht bequem eingesehen werden und die Größe des Repositories steigt unter Umständen an.

3.6 Weiterführende Links

- `man git`, `man gittutorial`
- Pro Git: <http://git-scm.com/book>
- Git - Documentation: <http://git-scm.com/doc>
- Git Immersion: <http://gitimmersion.com/>
- Easy Version Control with Git: <http://net.tutsplus.com/tutorials/other/easy-version-control-with-git/>
- Git (Wikipedia): [http://en.wikipedia.org/wiki/Git_\(software\)](http://en.wikipedia.org/wiki/Git_(software))
- Mögliche Nachteile von Git: <http://youtu.be/CDeG4S-mJts>

4 Python



<http://python.org>

Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.

4.1 IPython

Python selbst kommt mit einer interaktiven Kommandozeile (genauer: einer REPL¹-Umgebung). Um einen Einblick in die Sprache zu erhalten, ist diese eigentlich vollkommen ausreichend. Sie bietet die Möglichkeit, interaktiv mit der Sprache in Kontakt zu kommen. IPython erlaubt zusätzlich, auf die Shell zuzugreifen und interaktive Sitzungen zu speichern. Gestartet wird IPython auf der Kommandozeile mit

```
ipython3
```

und meldet sich (zum Beispiel hier unter OS X mit Python 3.2.3) mit der Ausgabe

```
Python 3.2.3 (default, Apr 13 2012, 00:15:25)
Type "copyright", "credits" or "license" for more information.
```

```
IPython 0.13 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.
```

In [1]:

Um IPython zu beenden gibt es die Befehle `exit`, `exit()`, `quit`, `quit()` und natürlich `Ctrl-D`.

¹read-eval-print loop

4.2 Syntax

Grundsätzlich ist die Python-Syntax sehr einfach und intuitiv. In vielen Punkten erinnert sie eher an eine Scriptsprache.

Blöcke Durch Einrückung! 4 Leerzeichen (/1 Tab)

Semikolons Gibt es prinzipiell Sind am Zeilenende aber nicht notwendig

??? In einer interaktiven Sitzung (z.B. IPython) können zum Einstieg einfache mathematische Berechnungen durchgeführt werden:

```
In [1]: 1 + 2
Out[1]: 3
```

```
In [2]: 1 * 2
Out[2]: 2
```

4.2.1 Variablen

Natürlich gibt es Variablen. Dynamische Typisierung Keine explizite Deklaration Ihre Verwendung ist denkbar einfach:

```
In [3]: a = 2
```

```
In [4]: a
Out[4]: 2
```

```
In [5]: b = 1 + 1
```

```
In [6]: b
Out[6]: 2
```

Variablenamen können Buchstaben, Zahlen und Unterstriche enthalten, dürfen jedoch nicht mit einer Ziffer beginnen.

4.2.2 Datenstrukturen/-typen

- bool (True, False)
- int, float, long, complex
- string ('foo', "bar")
- Iteratoren, Generatoren, Sequenzen, ...

Praktische Typen

- () Tupel
- [] Liste
- { } Dictionary

Zum Beispiel

```
In [9]: cities = ['Dortmund', 'Hamburg', 'Berlin']
```

```
In [10]: cities[0]
```

```
Out[10]: 'Dortmund'
```

Mehr Beispiele

```
In [1]: teams = {
...:     'BVB': "BV Borussia Dortmund 09",
...:     'S04': "FC Schalke 04",
...:     'FCB': "FC Bayern München"
...: }
```

```
In [2]: teams['BVB']
```

```
Out[2]: 'BV Borussia Dortmund 09'
```

4.2.3 Operatoren

Wie aus dem obigen Beispiel hervorgeht, gibt es in Python **mathematische Operatoren**. In Tabelle 4.1 sind diese dargestellt.

Tabelle 4.1: Mathematische Operatoren

Op.	Funktion	Beispiel	Ergebnis
+	Addition	1 + 2	3
-	Subtraktion	1 - 2	-1
*	Multiplikation	1 * 2	2
/	Division	1 / 2	0.5
%	Modulo	11 % 2	1
**	Exponent	2**3	8
//	Ganzzahldivision	5 // 2	2

Zuweisungsoperatoren setzen sich aus einem mathematischen Operator und einem Gleichheitszeichen zusammen. Mit ihnen kann gleichzeitig eine Berechnung und eine Zuweisung durchgeführt werden, zum Beispiel:

```
In [7]: a = 2
```

```
In [7]: a *= 3
```

```
In [8]: a
```

```
Out[8]: 6
```

Mit Hilfe von **Vergleichsoperatoren** (siehe Tabelle 4.2) lassen sich Vergleiche durchführen.

Tabelle 4.2: Vergleichsoperatoren

Op.	Funktion	Beispiel	Ergebnis
==	Gleich	1 == 1	True
!=	Ungleich	2 != 1	True
>	Größer	2 > 1	True
<	Kleiner	1 < 2	True
>=	Größer oder gleich	2 >= 1	True
<=	Kleiner oder gleich	1 <= 2	True

Logische Operatoren

`and`, `or`, `not`

Identitätsoperatoren

`is`, `is not`

Operatoren für Sequenzen

`in`, `not in`

4.2.4 Dateien

python open file write range len string format join list append index dict sorted decimal
math floor ceil convert str int float

4.3 Bibliotheken

4.3.1 NumPy

NumPy ist eine Bibliothek für Python, die diverse Features unterstützt, die wissenschaftliches Arbeiten deutlich erleichtern. Einige davon sind

- ein n -dimensionales Array-Objekt

- ausgeklügelte, durchdachte Funktionen für Optimierung, Integration und Interpolation
- die Möglichkeit C/C++- und Fortran-Code zu integrieren
- Tools für Lineare Algebra, Fouriertransformation, Statistik
- beliebige eigene Datentypen können definiert werden

4.3.2 SciPy

SciPy ist, ähnlich wie NumPy, eine Bibliothek, die wissenschaftliches Arbeiten erleichtert. Insbesondere sind in dem Paket verschiedene Konstanten gespeichert, die im Praktikum oft gebraucht werden. SciPy ist eine Erweiterung von NumPy, die vor allem vom NumPy-Array profitiert. Es besitzt ebenfalls viele benutzerfreundliche und effiziente numerische Algorithmen für Integration und Optimierung.

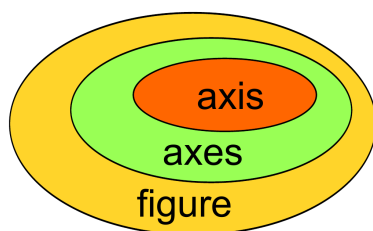
4.3.3 matplotlib

Bei matplotlib handelt es sich um eine Bibliothek für Plots.

matplotlib tries to make easy things easy and hard things possible.

Es dient dazu Plots, auch polare, sowie Fehlerbalken und noch vieles mehr zu Plotten. Es bietet sehr viele Einstellungsmöglichkeiten mit denen man den Plot anpassen kann. matplotlib bietet eine objektorientierte API, die erlaubt die Plots in Anwendungen einzubetten die generische GUI-Toolkits wie Qt oder GTK benutzen. Es gibt ebenfalls PyLab, das die Verwendung in IPython erleichtert.

Objektorientiertes Interface



- figure: Leinwand, auf die Plots projiziert werden
- axes: enthält Plot, der beliebig in der figure platziert werden kann
- axis: Koordinatenachsen

4.4 Python 2

Falls man Python 2 verwendet, sollte jede `.py`-Datei so anfangen:

```
# coding=utf-8
from __future__ import division, print_function, unicode_literals
```

Dann funktionieren `Division`, `print` und `Unicode` wie in Python 3. Man sollte dann auch immer `python2` und `ipython2` aufrufen.

4.5 Weiterführende Links

- Learn Python The Hard Way (Python 2): <http://learnpythonthehardway.org/>
- Dive Into Python 3: <http://www.diveintopython3.net/>
- Python v3 documentation: <http://docs.python.org/py3k/>
- PEP 8 -- Style Guide for Python Code: <http://www.python.org/dev/peps/pep-0008/>
- Tentative NumPy Tutorial: http://www.scipy.org/Tentative_NumPy_Tutorial
- NumPy and SciPy Documentation: <http://docs.scipy.org/doc/>
- matplotlib Documentation: <http://matplotlib.org/1.2.0/contents.html>
- matplotlib Gallery: <http://matplotlib.org/1.2.0/gallery.html>
- Python Uncertainties package: <http://packages.python.org/uncertainties/>
- SymPy: <http://sympy.org/en/index.html>
- matplotlib tutorial: <http://www.loria.fr/~rougier/teaching/matplotlib/>
- Python Scientific Lecture Notes: <http://scipy-lectures.github.com/>
- PyCon 2012: <http://pyvideo.org/category/17/pycon-us-2012>
 - Plotting with matplotlib: <http://pyvideo.org/video/617/plotting-with-matplotlib>
 - IPython: Python at your fingertips: <http://pyvideo.org/video/640/ipython-python-at-your-fingertips>
 - IPython in-depth: high-productivity interactive and parallel python: <http://pyvideo.org/video/605/ipython-in-depth-high-productivity-interactive-a>
 - Python for data lovers: explore it, analyze it, map it: <http://pyvideo.org/video/676/python-for-data-lovers-explore-it-analyze-it-m>

-
- Sage: Open Source Math in Python: <http://pyvideo.org/video/652/sage-open-source-math-in-python>
 - SciPy 2012: http://pyvideo.org/category/20/scipy_2012
 - Introduction to NumPy and matplotlib: <http://pyvideo.org/video/1190/introduction-to-numpy-and-matplotlib>
 - Advanced matplotlib: <http://pyvideo.org/video/1344/advanced-matplotlib>
 - IPython: tools for the entire lifecycle of research computing: <http://pyvideo.org/video/1221/ipython-tools-for-the-entire-lifecycle-of-research>
 - A tale of four libraries: <http://pyvideo.org/video/1211/a-tale-of-four-libraries>
 - SciPy 2011: <http://archive.org/search.php?query=subject%3A%22SciPy2011%22>
 - Protokolle (manchmal nicht sehr schön...)
 - Python-Bibliotheken, L^AT_EX-Header: <https://bitbucket.org/ibab/powertools>
 - Christian, Ismo (2011): <https://bitbucket.org/itoijala/ap-protokolle11>
 - Christian, Ismo (2012): <https://bitbucket.org/itoijala/ap-protokolle12>
 - Igor, Peter (2011):
 - Igor, Peter (2012):

5 L^AT_EX

5.1 useful packages

5.1.1 must

- `fixltx2e`
- `babel` [`ngerman`]
- `translator` [`ngerman`]
- `biblatex` [`backend=biber,sortlocale=de_DE.UTF-8`]
- `amsmath` [`sumlimits,intlimits,namelimits`]
- `mathtools`
- `fontspec` [`no-math`]
- `unicode-math` [`math-style=ISO,bold-style=ISO,sans-style=italic,nabla=upright,partial=`
- `mhchem` [`version=3`]
- `siunitx`
- `graphicx`
- `array`
- `booktabs`
- `hyperref` [`unicode=true,pdfcreator=,pdfproducer=`]

5.1.2 should

- `letltxmacro`
- `luatexbase`
- `xcolor`
- `amssymb`
- `IEEEtrantools` [`retainorgcmds`]

- tensor
- pdfscape
- placeins [above,below,section]
- caption [margin=10pt,font=small,labelfont=bf]

5.1.3 can

- expl3
- xparse
- metalogo
- tocbibind [notbib,nottoc]
- multirow
- csquotes [strict]
- minted
- tikz
- hyphenat
- extdash [shortcuts]