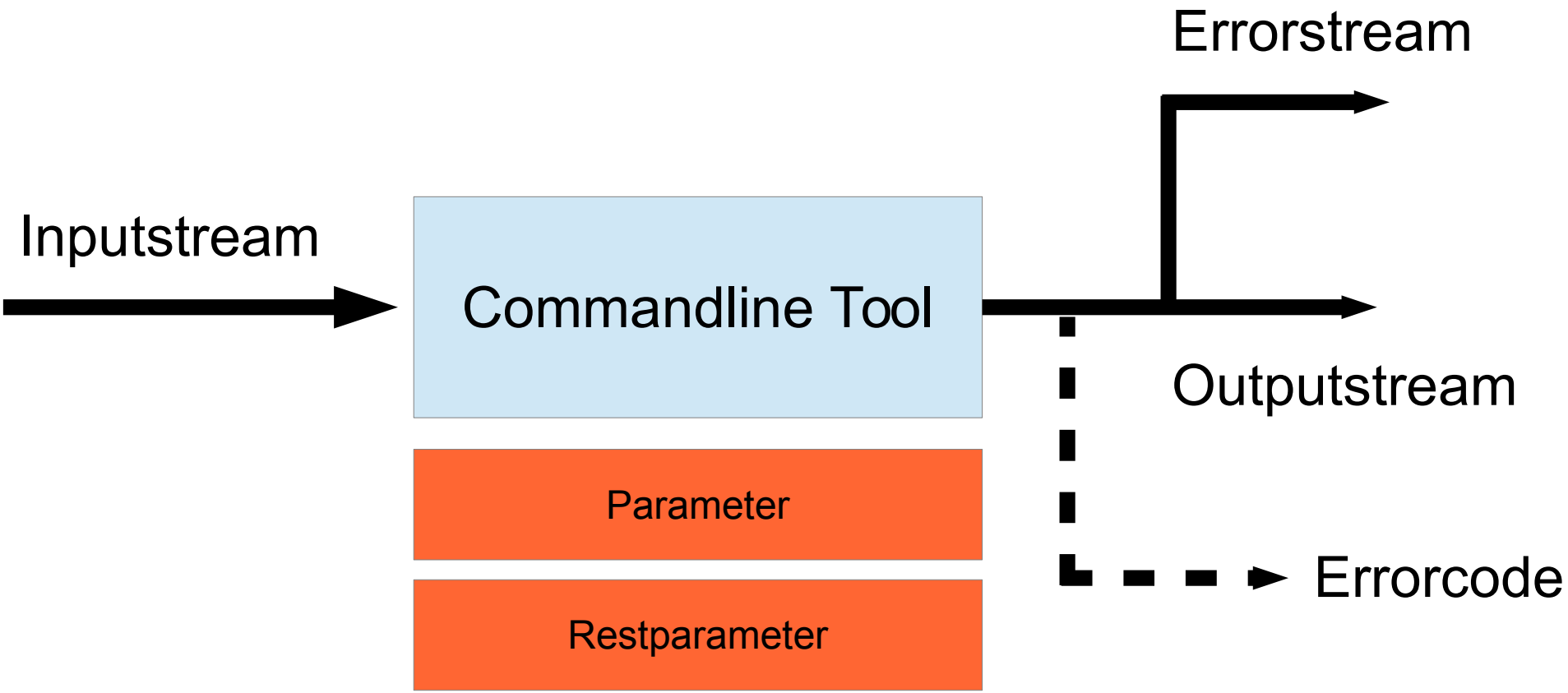
# Kommandozeilenprogramme

Marvin Meusel, Bertram Vogel

Lehrstuhl für Bioinformatik
Friedrich Schiller Universität Jena
10.11.2014


Errorstream
Inputstream
Commandline Tool
Outputstream
Parameter
Errorcode
Restparameter

# Errorcode

- Jedes Programm gibt einen Integer aus, der angibt, ob das Programm mit einem Fehler abgestürzt ist
- Rückgabewert 0 bedeutet: kein Fehler
- Ansonsten kann man jedem Zahlenwert einen Fehler zuordnen
- In Java: System.exit(errorcode)

# Streams

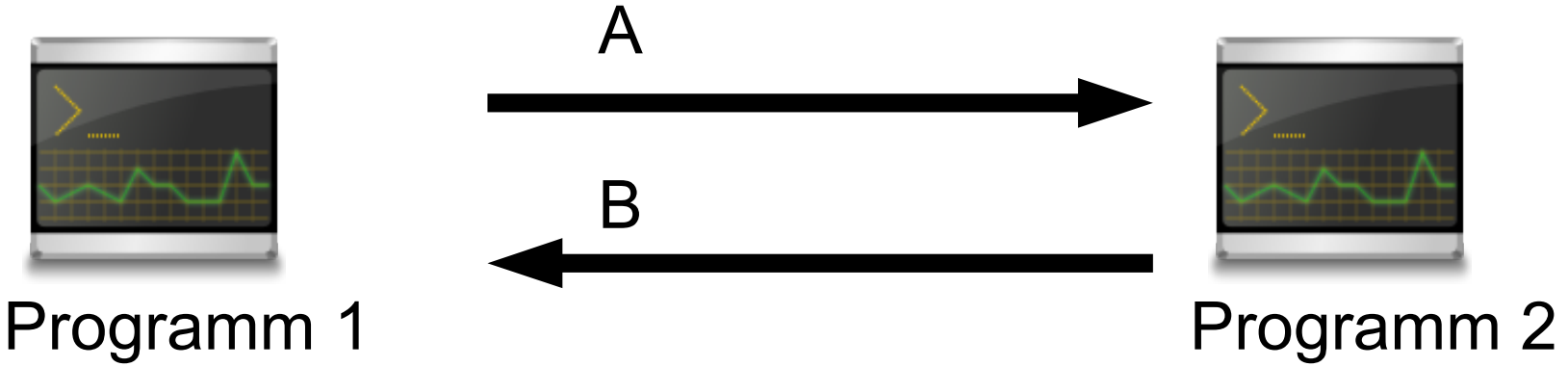
- “Fluss” von Bytes
- Anders als Array: Normalerweise kein wahlfreier Zugriff
- Stattdessen muss ein Stream  kontinuierlich ausgelesen/geschrieben werden
- Potentiell “unendlich”
- Beispiel: Streaming von Filmen auf Youtube

# Streams

- Datei
- Netzwerk
- Nutzereingabe
- Mikrofon, Webcam, ...
- Interprozesskommunikation
- ...

# Streams

- Inputstream:
  - Der Stream, der vom Programm eingelesen wird
- Outputstream
  - Der Stream, in den das Programm hineinschreibt



- Jedes Programm kennt drei Standardstreams:
  - Inputstream (die Eingabe) **System.in**
  - Outputstream (die Ausgabe) **System.out**
  - Errorstream (Fehlermeldungen und Warnungen) **System.err**

- in Konsolenanwendungen werden Outputstream und Errorstream standardmäßig “vermischt”.
- Eclipse und Intellij heben den Errorstream üblicherweise **rot** hervor
- System.out.println(...) schreibt in die Standardausgabe

# Streams und Reader/Writer

- Stream = Folge von Bytes (byte)
  - z.B. System.in
- Meistens will man aber nicht auf Bytes arbeiten, sondern auf Zeichen → Streams müssen vor der Verwendung in einen Reader oder Writer gekapselt werden
  - Reader/Writer = Folge von Zeichen (char)
  - z.B. InputStreamReader r = new InputStreamReader(System.in)
- Oft sinnvoll: nicht jedes Zeichen einzeln lesen, sondern mehrere auf einmal: BufferedReader/BufferedWriter
  - z.B. BufferedReader r = new BufferedReader(new InputStreamReader(in));

# Streams und Dateien

- In Java sind die Klassen **FileInputStream** (Datei lesen) und **FileOutputStream** (Datei schreiben) die Schnittstelle zwischen Stream und Datei
- In der Konsole gibt es die **Redirection** Operatoren < und >

wc -l < eingabe.txt > ausgabe.txt

  - liest eingabe.txt ein, zählt die Anzahl an Zeilen, schreibt das Ergebnis in ausgabe.txt

# Streams und Dateien

- 2> ist der Redirect Operator für den Errorstream

  *someprogram* < someinput.txt 2> logfile.txt

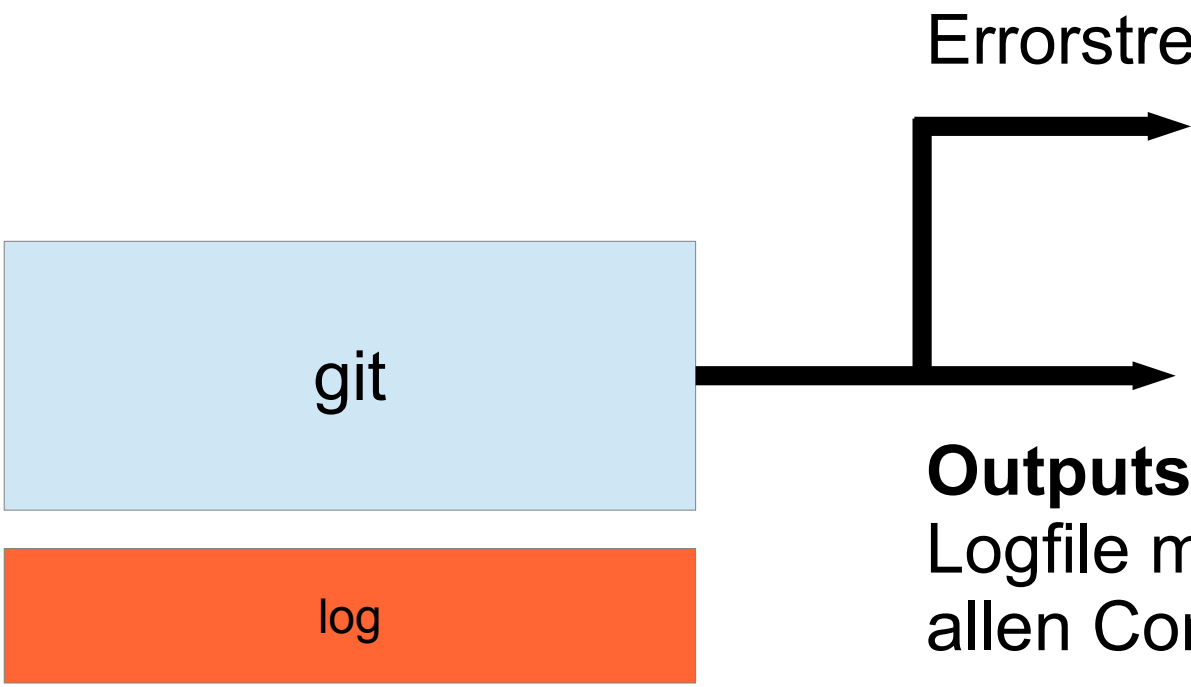
- >> fügt den Output ans Ende der Datei an, statt die Datei zu überschreiben
- der Pipe-Operator | ermöglicht das Verbinden von Programmen:

  *someprogram* | *otherprogram*

  - setzt den Outputstream von *someprogram* zum Inputstream von *otherprogram*

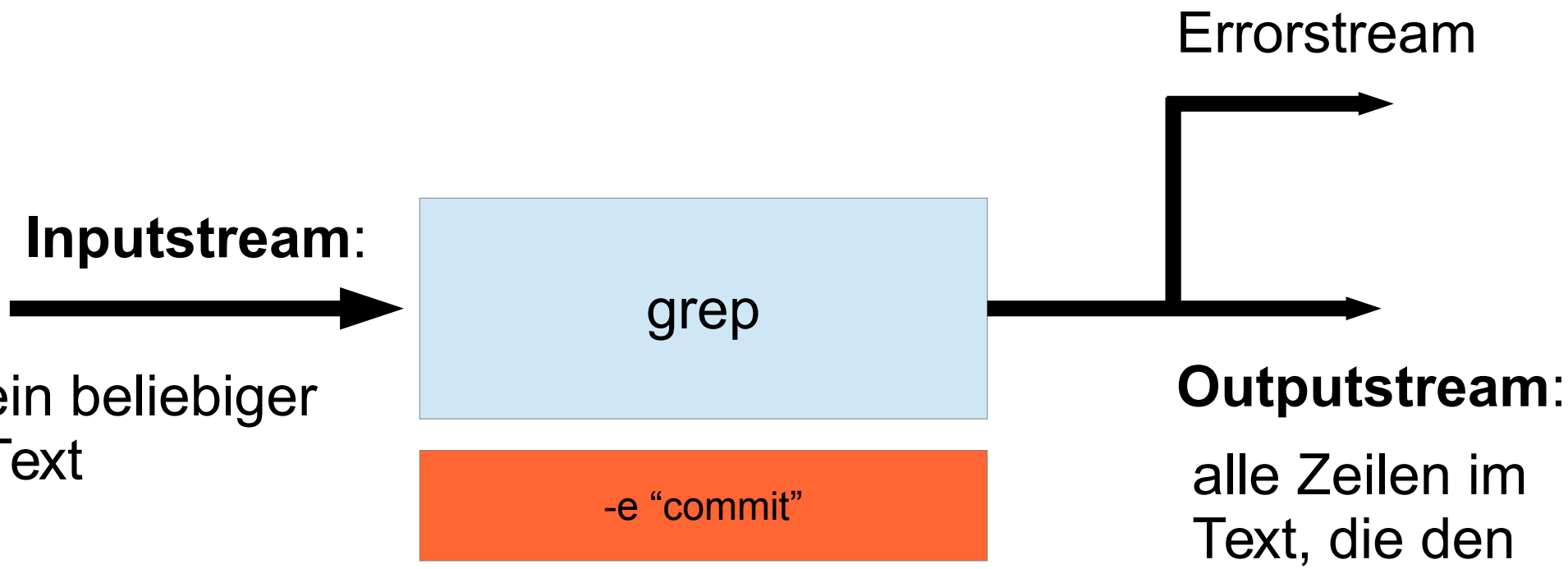
```
git log | grep -e “commit” | wc -l >> gitcommits.txt
```

dieses kleine Script zählt wie viele Commits ihr in eurem Repository gemacht habt und schreibt diese Zahl in eine Datei



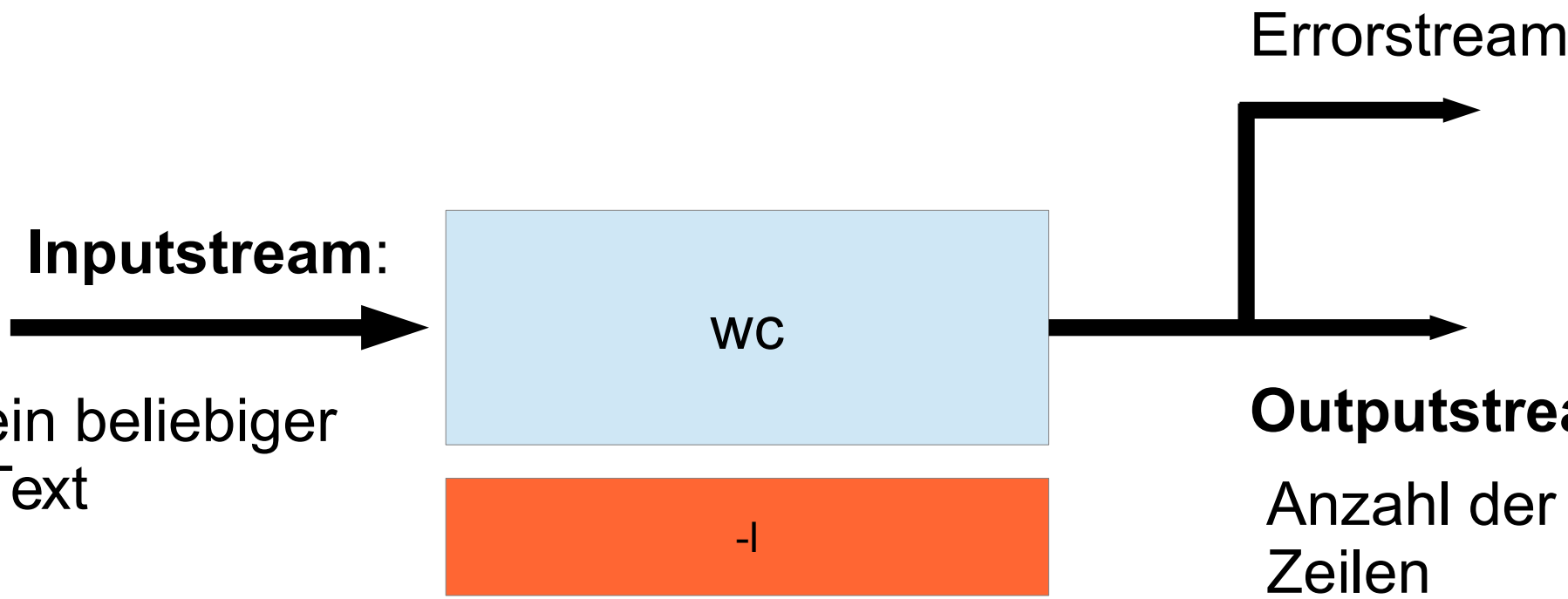

**Outputstream**:
Logfile mit
allen Commits

```
commit 0aad1e012ad2db5b298b1b680f7fcb6b97b5d49f
Author: Kai Dührkop <kai.duehrkop@uni-jena.de>
Date:   Tue Nov 26 14:16:32 2013 +0100

    profiles

commit 4e86dc28cd411daafab992ec8982ae547e0c02ac
Merge: 02bb05a b4c38ae
Author: Kai Dührkop <kai.duehrkop@uni-jena.de>
Date:   Mon Nov 4 11:43:09 2013 +0100

    Merge branch 'hotfix_dynamicilp'
    Now, DP solver is used if ilp solver is not provided

commit 02bb05a48cb4c7ba6e25be7c2585fdd2cad1a1c7
Merge: 130c803 f66e34e
Author: Kai Dührkop <kai.duehrkop@uni-jena.de>
Date:   Mon Nov 4 11:42:52 2013 +0100

    uptodate

commit b4c38ae3ab11d49e3596bef613a86365eae6fd6d
Merge: 1c9505c f66e34e
```




```
commit 0aad1e012ad2db5b298b1b680f7fcb6b97b5d49f
Author: Kai Dührkop <kai.duehrkop@uni-jena.de>
Date:   Tue Nov 26 14:16:32 2013 +0100

    profiles

commit 4e86dc28cd411daafab992ec8982ae547e0c02ac
Merge: 02bb05a b4c38ae
Author: Kai Dührkop <kai.duehrkop@uni-jena.de>
Date:   Mon Nov 4 11:43:09 2013 +0100

    Merge branch 'hotfix_dynamicilp'
    Now, DP solver is used if ilp solver is not provided

commit 02bb05a48cb4c7ba6e25be7c2585fdd2cad1a1c7
Merge: 130c803 f66e34e
Author: Kai Dührkop <kai.duehrkop@uni-jena.de>
Date:   Mon Nov 4 11:42:52 2013 +0100

    uptodate

commit b4c38ae3ab11d49e3596bef613a86365eae6fd6d
Merge: 1c9505c f66e34e
```

```
$ git log | grep -e "commit"
commit e4fa07501d954b013c045eaadbee362f0433a5a9
commit b1cdebbdd966e29c61f205b42837a6f56ce3024a
commit 678d99cd4de9ab31524ff7f53cd59170855f7771
commit 61aad190ed75c3b2b4bbd1f15178adc464ebd200
commit 31d8582a1267385ebed435651ba635b39879d643
commit 91982466600e28ebcb5c4bfd24288145943ca1fe
commit f3268ed6c830be582580eaafd7dc3b393091ce22
commit 0aad1e012ad2db5b298b1b680f7fcb6b97b5d49f
```



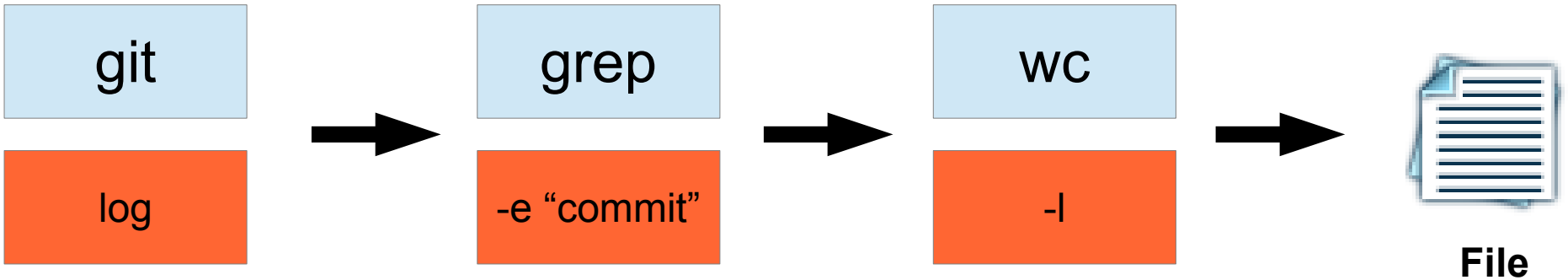

Option: Zähle Zeilen

```
$ git log | grep -e "commit" | wc -l
145
```

```
$ git log | grep -e "commit"
commit e4fa07501d954b013c045eaadbee362f0433a5a9
commit b1cdebbdd966e29c61f205b42837a6f56ce3024a
commit 678d99cd4de9ab31524ff7f53cd59170855f7771
commit 61aad190ed75c3b2b4bbd1f15178adc464ebd200
commit 31d8582a1267385ebed435651ba635b39879d643
commit 91982466600e28ebcb5c4bfd24288145943ca1fe
commit f3268ed6c830be582580eaafd7dc3b393091ce22
commit 0aad1e012ad2db5b298b1b680f7fcb6b97b5d49f
```

**git** log | **grep** -e “commit” | **wc** -l >> gitcommits.txt

```
$ git log | grep -e "commit"
commit e4fa07501d954b013c045eaadbee362f0433a5a9
commit b1cdebbdd966e29c61f205b42837a6f56ce3024a
commit 678d99cd4de9ab31524ff7f53cd59170855f7771
commit 61aad190ed75c3b2b4bbd1f15178adc464ebd200
commit 31d8582a1267385ebed435651ba635b39879d643
commit 91982466600e28ebcb5c4bfd24288145943ca1fe
commit f3268ed6c830be582580eaafd7dc3b393091ce22
commit 0aad1e012ad2db5b298b1b680f7fcb6b97b5d49f
```




```
commit 0aad1e012ad2db5b298b1b680f7fcb6b97b5d49f
Author: Kai Dührkop <kai.duehrkop@uni-jena.de>
Date:   Tue Nov 26 14:16:32 2013 +0100

    profiles

commit 4e86dc28cd411daafab992ec8982ae547e0c02ac
Merge: 02bb05a b4c38ae
Author: Kai Dührkop <kai.duehrkop@uni-jena.de>
Date:   Mon Nov 4 11:43:09 2013 +0100

    Merge branch 'hotfix_dynamicilp'
    Now, DP solver is used if ilp solver is not provided

commit 02bb05a48cb4c7ba6e25be7c2585fdd2cad1a1c7
Merge: 130c803 f66e34e
Author: Kai Dührkop <kai.duehrkop@uni-jena.de>
Date:   Mon Nov 4 11:42:52 2013 +0100

    uptodate

commit b4c38ae3ab11d49e3596bef613a86365eae6fd6d
Merge: 1c9505c f66e34e
```

```
$ git log | grep -e "commit" | wc -l
145
```

# Kurzes Fazit

- Kommandozeilenprogramme ermöglichen das Verknüfen verschiedener Programme über deren Input- und Outputstreams (solche verknüften Programme nennt man **Pipelines**)
- komplexe Aufgaben lassen sich automatisieren, in dem man für jede Einzelaufgabe ein eigenes, kleines Kommandozeilenprogramm nutzt und diese dann miteinander verknüft

# Parameter

- Nachteil von Streams: Wir haben nur 1 Eingabe und 2 Ausgabestreams
- Das reicht uns nicht!
- viele Programme brauchen mehrere Eingaben
- daher nehmen sie Parameter entgegen

# Parameter

- Jedes Programm erhält einen Array aus Strings als Parameter
- auf der Konsole werden diese Parameter nach dem Programmnamen eingeben und mit Leerzeichen voneinander getrennt
- braucht man das Leerzeichen als Zeichen in einem Parameter, so muss man dieses escapen (mit Backslash) oder den Parameter in Anführungszeichen schreiben

# Parameter

**myprogram** parameter1 parameter2 parameter3

**myprogram** “parameter 1”

**myprogram** parameter\ 1

# Arten von Parametern

- darüber hinaus gibt es Konventionen, wie Parameter für ein Programm auszusehen haben
- dies erleichtert die Bedienung des Programms für Außenstehende!
- verschiedene Programme sind auf die gleiche Weise zu bedienen
  - Bsp: cp -r und rm -r
  - In beiden Fällen bedeutet das -r “rekursiv”

# Boolscher Parameter

```
search --count
```

Es ist nur entscheidend, ob der Parameter gesetzt ist (true) oder nicht (false)

```
search -c
```

Viele Parameter erlauben “Kurzformen”. Die “Langform” fängt in der Regel mit zwei Bindestrichen an, die Kurzform besteht nur aus einem Buchstaben und fängt mit einem Bindestrich an

# Boolscher Parameter

```
search --count --ignore --measure
```

```
search -c -i -m
```

```
search -cim
```

Manche Programme erlauben das Zusammenfassen von boolschen Parametern.

# Value Parameter

```
search --algorithm=KMP --frame=3
```

```
Search --algorithm KMP --frame 3
```

```
Search --algorithmKMP --frame3
```

Letztere Form wird nicht von allen Programmen unterstützt.

# Value Parameter

```
search --algorithm=”Knuth Morris Pratt”
```

Wenn Parameter Leerzeichen enthält, muss dieser escaped oder in Anführungszeichen geschrieben werden

# Rest-Parameter

```
search -p “ACAG” somedir/*.fasta
```

- Viele Programme akzeptieren am Ende ihrer Parameterliste eine Folge von Files (oder beliebigen anderen Strings)
- diese können entweder nacheinander eingetippt, oder über Bash-Wildcards erzeugt werden: *.fasta
- jedes File ist ein Parameter in der Parameterliste

# Subcommands

```
search count
```

- Widersprechen eigentlich dem Prinzip, Kommandozeilenprogramme klein und einfach zu halten
- Beispiel: git log, git clone usw.

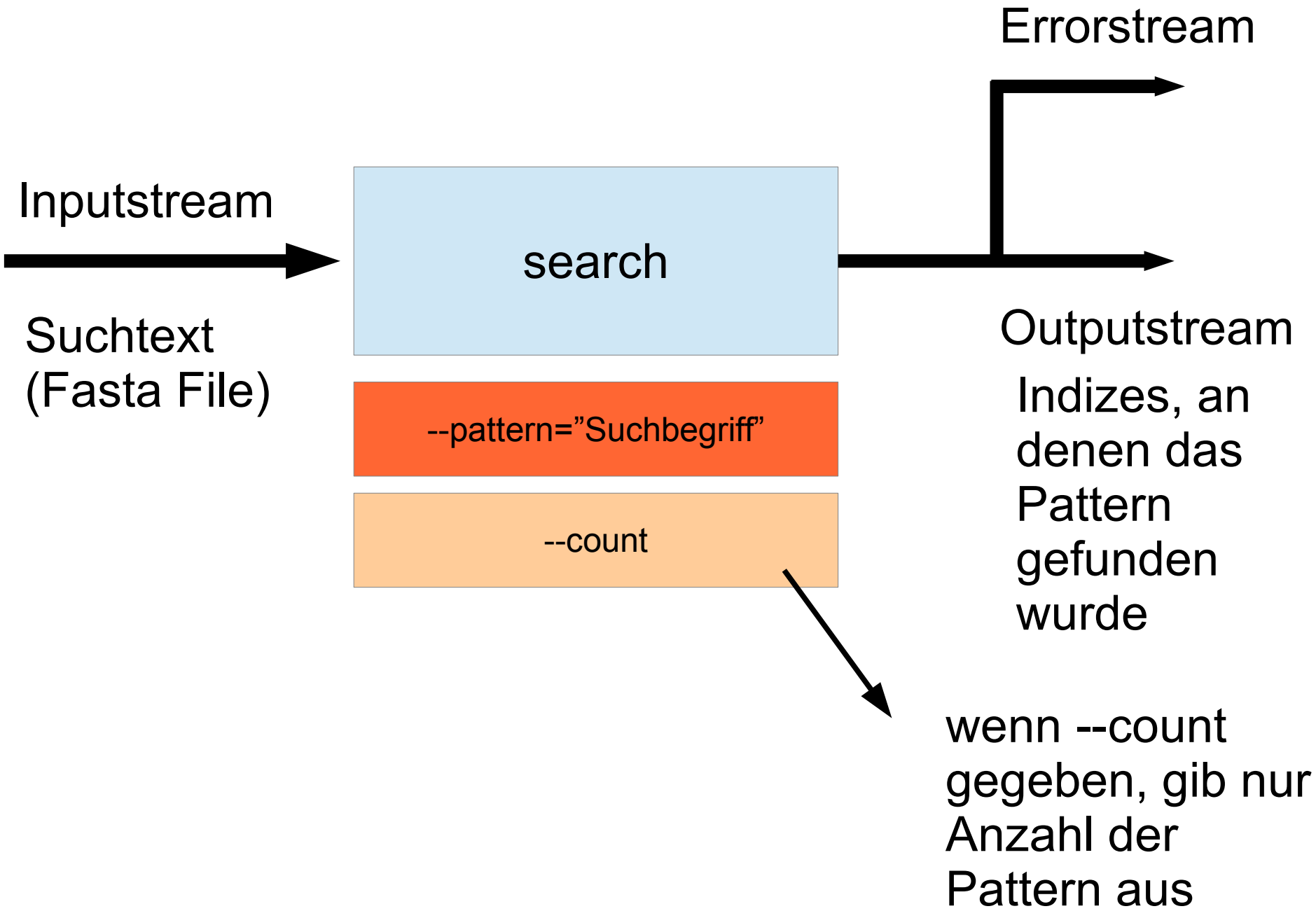
```
search --searchoption count --countoption
```

Parameter können sinnvoll geordnet und so als allgemeine Parameter und Subcommand Parameter unterteilt werden

- Das alles sind nur Konventionen!
- Java-Tools (javac, java, jar, ...) nutzen z.B. immer nur ein Bindestrich für ihre Parameter

- Übliche (allgemeine) Parameter:
  - -v und --verbose für: Informative Ausgaben
  - -h und --help für: Gib Dokumentation zu Kommandozeilenprogramm aus
  - --version für: Gib Programmversion aus

# Beispiel: Exakte Suche



# Beispiel: Exakte Suche

```
search < sequence.fasta -p ACCGACTA -c
```

```
search < sequence.fasta -p ACCGACTA > indizes.txt
```

# Umsetzung in Java

- Einfache Kommandozeilentools kann man leicht selbst schreiben
- Alternativ gibt es viele Bibliotheken, die einem diese Arbeit erleichtern.
- Für java: simpleopts, jewelcli, args4j, JSAP, ...

```
enum Algorithm {NAIVE, KMP, BM}

public static void main(String[] args) {

  Algorithm algorithm = Algorithm.NAIVE;
  boolean count = false;

  for (int i = 0; i < args.length; ++i) {
    String opt = args[i];
    if (opt.equals("-h") || opt.equals("--help")) {
      printHelp();
    } else if (opt.equals("--algorithm") || opt.equals("-a")) {
      String value = args[++i];
      try {
        algorithm = Algorithm.valueOf(value.toUpperCase());
      } catch (IllegalArgumentException e) {
        System.err.println("Unknown algorithm " + value);
        System.err.println("Use one of NAIVE, KMP or BM");
      }
    } else if (opt.equals("-c") || opt.equals("--count")) {
      count = true;
    } else if ...
```

# Jewelcli

- http://jewelcli.lexicalscope.com/
- enthält Tutorial mit vielen Beispielen
- Man definiert seine Kommandozeilenoptionen als Getter Methoden eines Interfaces mit Annotationen
- Zur Laufzeit wird dann eine Implementation des Interfaces generiert, welche die geparsten Parameter beinhaltet

```
public class CLI {

    enum Algorithm {NAIVE, KMP, BM}

    interface Options {

        @Option(shortName = "a", description = "Algortihm to use. "
                + "Either NAIVE, KMP or BM.", defaultValue = "NAIVE")
        public Algorithm getAlgorithm();

        @Option(shortName = "p", description = "Pattern to search.")
        public String getPattern();

        @Option(shortName = "c", description = "Only report number of matches")
        public boolean isCount();

        @Option(shortName = "h", helpRequest = true)
        public boolean isHelp();

    }
}
```

- Value Parameter mit Default Value sind optional
- der lange Name ist immer der Methodenname ohne get und set
- der kurze Name wird mit shortName übergeben
- dabei werden die Konventionen eingehalten: --algorithm -a
- description ist der Text, der bei der Hilfe angezeigt wird

```
public static void main(String[] args) {

    Options options;
    try {
        Options = CliFactory.parseArguments(Options.class, args);
    } catch (ArgumentValidationException e) {
        System.err.println(e.getMessage());
        System.exit(1);
        return;
    }

    BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
    StringBuilder buffer = new StringBuilder();
    String line;
    try {
        while((line = input.readLine()) != null) buffer.append(line).append('\n');
    } catch (IOException e) {
        System.err.println("Error while reading input file: " + e.getMessage());
        System.exit(1);
    }
    String searchText = buffer.toString();
    String pattern = opuions.getPattern();
    boolean count = options.isCount();
    switch (options.getAlgorithm()) {
        case NAIVE: searchNaive(searchText, pattern, count); break;
        case KMP: searchKMP(searchText, pattern, count); break;
        case BM: searchBM(searchText, pattern, count); break;
    }
}
```

zeigt Hilfetext an

- wenn defaultToNull=true, ist der Parameter optional. Wird er nicht gesetzt, gibt die Methode null zurück
- @Unparsed wird für Restparameter genutzt
- exactly=2 bedeutet: Es sollen genau 2 zusätzliche Restparameter mitgegeben werden
- jewelcli bietet noch viele weitere Features an
- lest euch einfach mal das Tutorial dazu durch ;)