Ausgabe 03/2024

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

D: 4,90 EUR | CH: 9,80 CHF | A: 5,60 EUR | Benelux: 5,80 EUR

4 191978 304903 03

# Java aktuell

iJUG Verbund
www.ijug.eu

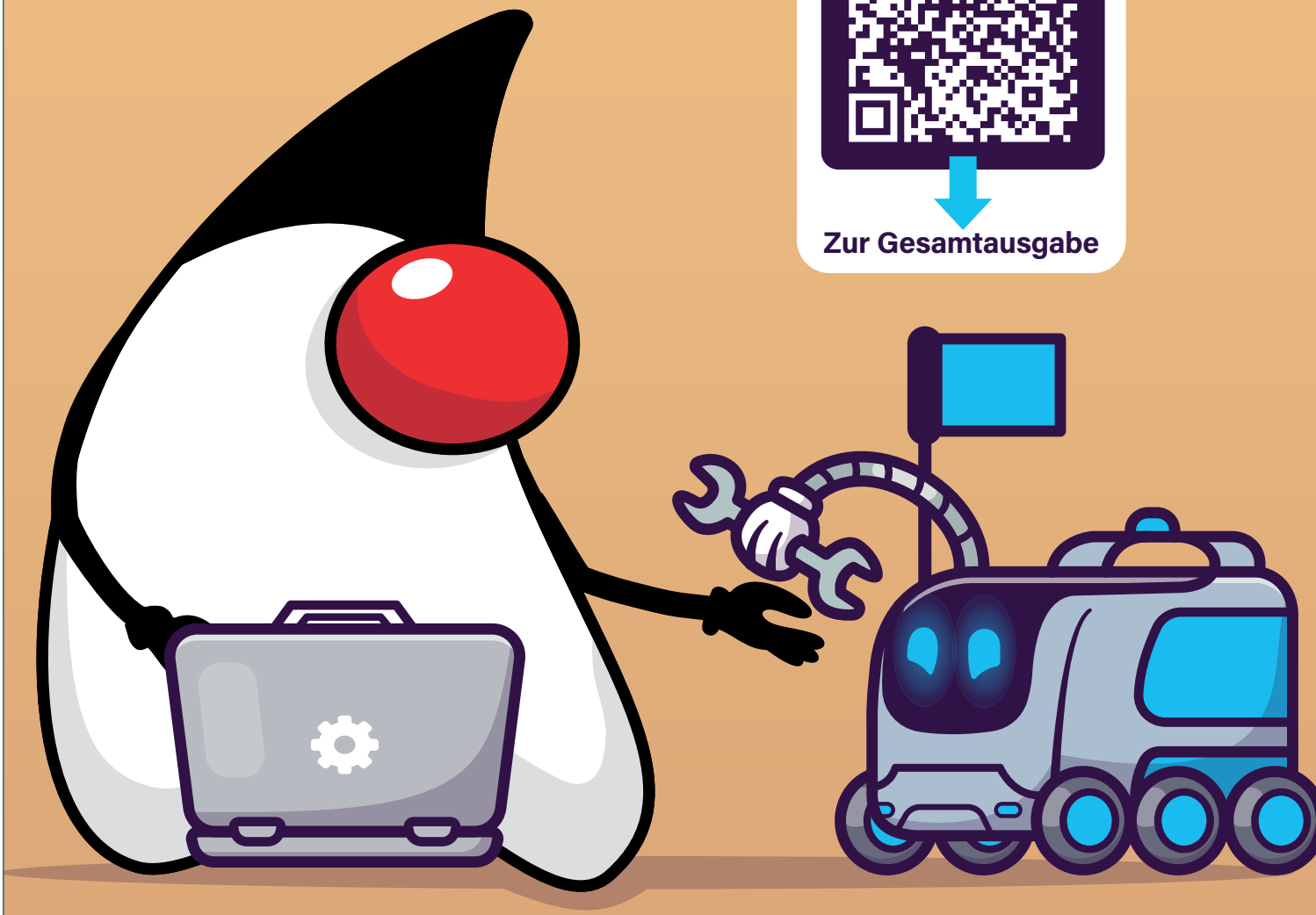
**Tools**
Shell, GitHub Actions mit Dependabot

**Testing**
Playwright Java, Test-First-Strategie

**Cloud Native**
Cloud-Native-Applikationen mit Spring Boot 3 und GraalVM

# TOOLS

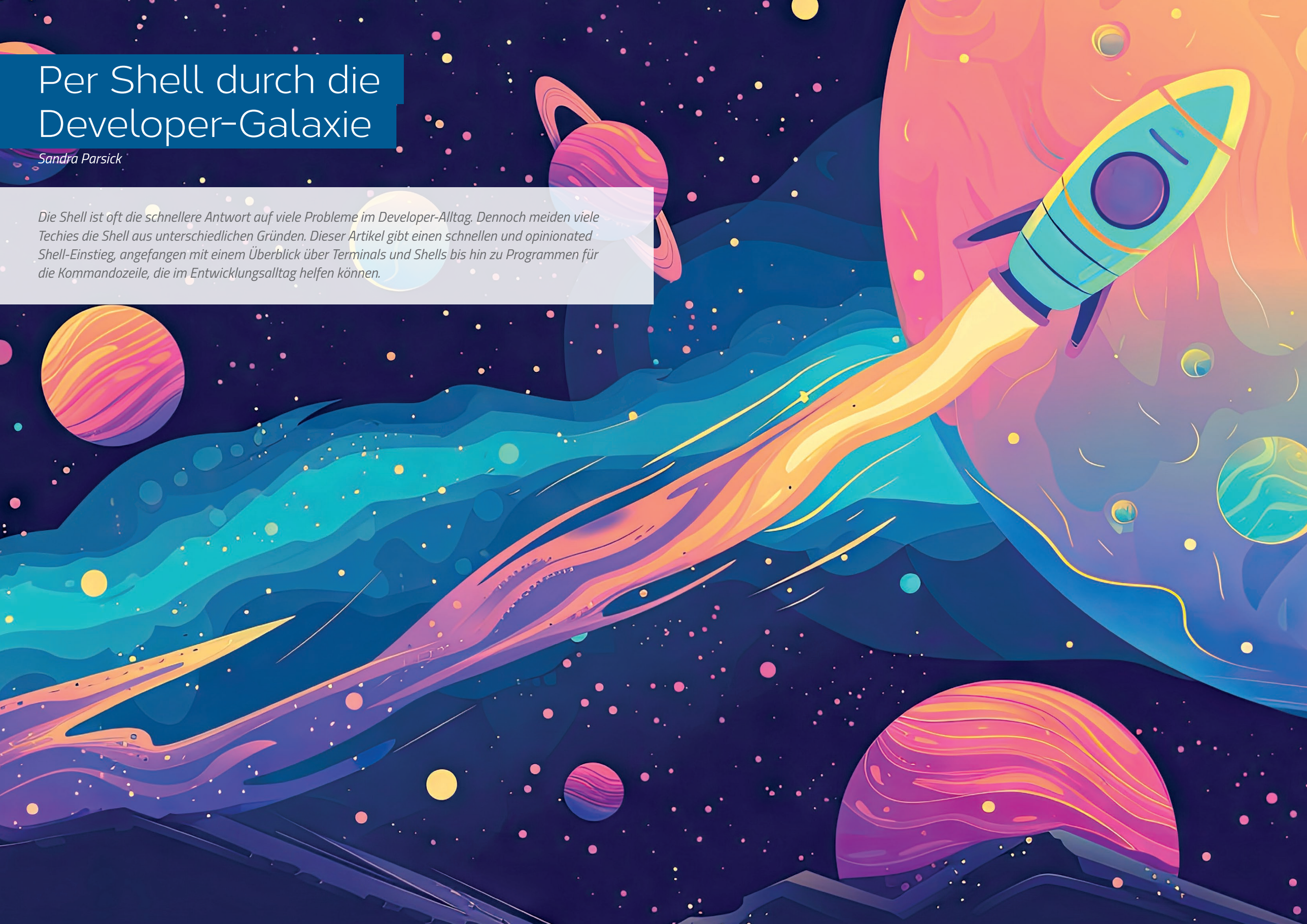
Zur Gesamtausgabe



# Per Shell durch die Developer-Galaxie

*Sandra Parsick*

*Die Shell ist oft die schnellere Antwort auf viele Probleme im Developer-Alltag. Dennoch meiden viele Techies die Shell aus unterschiedlichen Gründen. Dieser Artikel gibt einen schnellen und opinionated Shell-Einstieg, angefangen mit einem Überblick über Terminals und Shells bis hin zu Programmen für die Kommandozeile, die im Entwicklungsalltag helfen können.*



## Terminal, die Benutzeroberfläche

Allgemein gesprochen sind Terminals grafische Benutzeroberflächen, um textbasierte Programme, wie zum Beispiel eine *Shell*, zu verwenden. Strenggenommen, müssten Terminals „Terminalemulation" genannt werden, da sie die Funktion eines Computer-Terminals nachbilden *[1]*.

> Ein Computer-Terminal ist ein Endgerät zur Ein- und Ausgabe von Daten und dient als Benutzerschnittstelle, unter anderem für Großrechner. Heutzutage gibt es noch branchenspezifische Terminals wie zum Beispiel Kreditkartenterminals oder Geldautomaten.

Moderne Terminals bieten Komfort in der Bearbeitung der Eingabe (zum Beispiel durch einfaches Kopieren und Einfügen von Befehlen), Anordnung der Eingabefenster, Definition von Shortcuts, Plug-in-Erweiterung, Anpassung des Aussehens (Schriftgröße, Farbe usw.) und vieles mehr.

Mittlerweile gibt es für jedes Betriebssystem Terminals, die diese Funktionalitäten mitbringen. Wenn eine Entwicklerin auf einem Linux-System unterwegs ist, bringt die eingesetzte Desktop-Umgebung ihr eigenes Terminal mit. KDE hat oft *Konsole* *[2]* vorinstalliert, GNOME das *Gnome Terminal* *[3]*. Diese Terminals sind stark in der jeweiligen Desktop-Umgebung verankert. Das führt dazu, dass wenn ein Entwickler die Desktop-Umgebung wechseln will, aber seine Terminalsoftware aus der alten Umgebung weiterverwenden möchte, die Terminalsoftware eine Menge an Pakete von der alten Desktop-Umgebung benötigt und diese mitinstalliert werden. Mit *Tilix* *[4]* gibt es ein Terminal, dass von der Desktop-Umgebung unabhängig ist, aber dennoch die wichtigsten Funktionalitäten eines modernen Terminals mitbringt. Wenn eine Entwicklerin einen Mac benutzt, dann ist als Standard-Terminal *Terminal* *[5]* installiert. Viele Mac-User bevorzugen aber *iterm2* *[6]*. Auf Windows sah es lange Zeit nicht sehr gut im Hinblick auf komfortable Terminals aus. Mittlerweile bietet Microsoft allerdings das *Windows Terminal* *[7]* an.

Abseits dieser eher klassischen Terminals gibt es zum Beispiel mit *fig* *[8]*, *warp* *[9]* und *hyper* *[10]* Terminals mit neuen Ansätzen, die sich zum Teil wie eine IDE anfühlen. Die Auswahl ist groß und für jeden Geschmack ist etwas dabei.

## Shell, die Kommandozeile

Im allgemeinen Sprachgebrauch versteht eine Entwicklerin unter dem Begriff *Shell* die *Befehlszeilenschnittstelle im Textmodus* des Betriebssystems. Laut Definition der Wikipedia ist eine *Shell* die allgemeine Schnittstelle zum Betriebssystem *[11]*. Sie kann auch grafisch sein (GUI). Der Fokus des Artikels liegt aber auf der textbasierten Form.

Allgemein gesprochen, bietet eine Shell eine Umgebung an, in der Benutzer über eine Kommandozeile Befehle eingeben können, die ein Interpreter ausführt. Ein Befehl besteht aus dem Programmnamen gefolgt von *Argumenten* (Beispiel: *siehe Listing 1*).

```
ls -l /home
```

*Listing 1: Beispiel für einen Shell-Befehl*

*Argumente* können Optionen (beginnen oft mit - oder --) sein oder Eingabewerte für das Programm. Jede Shell bringt sogenannte Built-in-Befehle mit, das heißt, es handelt sich um Programme für die Kommandozeile, die mit der Shell installiert werden. Es können Programme für die Shell nachinstalliert werden, indem die Programme im sogenannten *Suchpfad* installiert werden.

> Der **Suchpfad** ist eine Liste mit Verzeichnissen, in der die Shell nach Programmen sucht. Der Wert der Shell-Variable PATH definiert diese Liste.

Auf unix-ähnlichen Systemen wie Linux und Mac kommen oft Shells wie *Bash* *[12]* und *Z shell (zsh)* *[13]* zum Einsatz. Windows bietet *CMD* und *Powershell* an. Dank dem Windows-Linux-Subsystem (WSL) können Entwickler auch die unix-basierten Shells unter Windows verwenden.

Unter Entwicklern ist die Shell *fish* *[14]* recht beliebt, da sie Funktionalitäten von Haus aus mitbringt, die bei anderen Shells nachinstalliert werden müssen, wie zum Beispiel Textautovervollständigung oder automatisierte Befehlvorschläge basierend auf den schon benutzten Befehlen. *Bash* und *zsh* sind recht ähnlich in der Benutzung und eine von beiden ist immer die Standardshell auf unix-ähnlichen Systemen. *zsh* bietet mehr Konfigurationsmöglichkeiten an als *Bash* und beim Scripting verhält sich *zsh* im Detail anders als Bash. Im Alltag macht sich der Unterschied kaum bemerkbar.

In der Community um die Shells haben sich Frameworks gebildet, um die Konfiguration der Shell besser zu verwalten (*oh-my-zsh* *[15]*, *oh-my-bash* *[16]*, *oh-my-posh* *[17]* um ein paar zu nennen). Darüber hinaus bieten sie an, die eigene Shell mit *alias*, Plug-ins und Themes zu erweitern. Wie sich dies im Alltag auswirkt, schauen wir uns am Beispiel *oh-my-zsh* an.

## Entwickleralltag mit oh-my-zsh

*oh-my-zsh* wird mithilfe von *curl* *[18]* und *git* *[19]* installiert. Die Installation sieht so aus, dass die Installationsroutine das Framework mit allen gebündelten Plug-ins und Themes unter `~/.oh-my-zsh` herunterlädt und die zsh-Konfigurationsdatei `.zshrc` neu angelegt, in der später definiert werden kann, welche Plug-ins und welche Themes benutzt werden.

Ein *alias* ist eine Möglichkeit in der Shell, mehrere Befehle oder einen Befehl mit mehreren Optionen zu einem neuen Befehl zusammenzufassen. *alias* ist eine Art Shortcut, *oh-my-zsh* bringt eine eigene Liste von *aliases* mit, die sich über Plug-ins erweitern lässt. Diese Alias helfen beispielsweise dabei, schneller im Dateisystem zu navigieren *(siehe Listing 2)*.

*oh-my-zsh* lässt sich mithilfe von Plug-ins erweitern. Es bringt eine Vielzahl an Plug-ins mit, die eine Entwicklerin in der Konfigurationsdatei aktivieren kann. Sie kann externe Plug-ins nachinstallieren *(siehe Listing 3)*.

Plug-ins sind wiederum eine Sammlung von *alias* und Shell-Funktionen zu einem bestimmten Werkzeug. Das Plug-in *git* bietet zum Beispiel fertige *alias* für Git an, um schneller Git-Befehle einzugeben und nützliche Shell-Funktionen rund um Git, um bestimmte Anwendungsfälle schneller auszuführen (Beispiel: *siehe Listing 4*).

Darüber hinaus bietet *oh-my-zsh* mithilfe von *Themes* an, das Aussehen der *Shell Prompt* anzupassen. Es bringt eine Vielzahl an Themes mit, die ein Entwickler in der Konfigurationsdatei aktivieren kann *(siehe Listing 5)*. Auch hier können externe Themes nachinstalliert werden.

> Mit Shell Prompt ist die Eingabeaufforderung in der Shell gemeint.

Die Theme-Konfiguration wirkt auf den ersten Blick wie eine Spielerei. Auf den zweiten Blick bewirkt die Anpassung der Prompt, dass man nützliche Informationen für die tägliche Arbeit direkt im Blick hat (zum Beispiel auf welchem Git Branch man sich aktuell befindet.) Themes wie *Starship* *[20]* (muss nachinstalliert werden) erweitern die Prompts um nützliche Informationen, so kann eine Entwicklerin beispielsweise sehen, welche Version von einer Runtime aktuell benutzt wird.

```
# Switching directory
<directory name>='cd <directory name>'
..='cd ..'
...='cd ../..'

# Create directory
md='mkdir -p'

# List directory content
l='ls -lah'
ll='ls -lh'
```

*Listing 2: Beispiel für alias-Definitionen in oh-my-zsh*

## Shellwerkzeuge, die den Developer-Alltag vereinfachen können

Die Shell kann ihr volles Potenzial ausspielen, wenn auch die richtigen Shell-Werkzeuge für die bevorstehende Aufgabe installiert sind. Doch welche Werkzeuge können wann und wie im Alltag helfen? Es folgt eine kleine Auflistung von nützlichen Werkzeugen.

## Werkzeugverwaltung vereinfachen

Je nach Projekt-Setup muss die Java-Entwicklerin mit unterschiedlichen Java- und Buildwerkzeugs-Versionen hantieren. Die einmalige

```
$ cat ~/.zshrc

# Which plugins would you like to load? (plugins can be found in ~/.oh-my-zsh/plugins/*)
# Custom plugins may be added to ~/.oh-my-zsh/custom/plugins/
# Example format: plugins=(rails git textmate ruby lighthouse)
# Add wisely, as too many plugins slow down shell startup.
plugins=(git mvn)
```

*Listing 3: Plug-in-Konfiguration in oh-my-zsh*

```
#alias
g='git'
gcl='git clone'
gl='git pull'
gsw='git switch'
gaa='git add -A'
gcmsg='git commit -m '
gp='git push'

#Functions
grename old new # reanmes branch old to new, including on the origin remote
gbda # deletes all merged branches
```

*Listing 4: Beispiele für git alias und Shell-Funktionen*

```
$ cat ~/.zshrc

# Set name of the theme to load.
# Look in ~/.oh-my-zsh/themes/
# Optionally, if you set this to "random", it'll load a random theme each
# time that oh-my-zsh is loaded.
#ZSH_THEME="agnoster"
ZSH_THEME="simple"
```

*Listing 5: Theme-Konfiguration in oh-my-zsh*

```
→ sdk list java # listet verfügbare Java Versionen auf (Ausschbnitt)
================================================================================
Available Java Versions for Linux 64bit
================================================================================
 Vendor        | Use | Version      | Dist    | Status     | Identifier
--------------------------------------------------------------------------------
 Temurin       | >>> | 21.0.1       | tem     | installed  | 21.0.1-tem
               |     | 17.0.9       | tem     | installed  | 17.0.9-tem
               |     | 11.0.21      | tem     | installed  | 11.0.21-tem
               |     | 8.0.392      | tem     | installed  | 8.0.392-tem


→ sdk install java 21.0.1-tem # installiert JDK Eclipse Temurin in Version 21.0.1
→ sdk default java 21.0.1-tem # setzt Eclipse Temurin in Version 21.0.1 als Default-JDK
→ sdk use java 17.0.9-tem # setzt Eclipse Temurin in Version 17.0.9 als JDK für die aktuelle Session
```

*Listing 6: Java-Versionen verwalten mit SDKMAN!*

```
→ cat .sdkmanrc
# Enable auto-env through the sdkman_auto_env config
# Add key=value pairs of SDKs to use below
java=17.0.9-tem
maven=3.9.6
```

*Listing 7: Beispiel für .sdkmanrc*

```
→ cd myproject
Using java version 17.0.9-tem in this shell.

Using maven version 3.9.6 in this shell.
```

*Listing 8: Beispiel für Autodetection*

Installation der Versionen ist in der Regel schnell erledigt, doch das Wechseln zwischen diesen ist oft recht mühselig. Dieses Problem möchte *SDKMAN!* *[21]* lösen.

Es bietet eine Schnittstelle an, um Werkzeuge aus dem JVM-Ökosystem (Java, Scala, Kotlin und Groovy, Ant, Gradle, Grails, Maven, SBT, Spark, Spring Boot, Vert.x und viele weitere) zu installieren und zu verwalten. Den Workflow für die Verwaltung von Java-Versionen zeigt *Listing 6*.

Nutzen alle im Team SDKMAN! Als Verwaltungswerkzeug, kann das Team die zu benutzende JDK-Version auch im Projekt definieren, indem sie eine `.sdkmanrc`-Datei im Root-Verzeichnis ablegen, die die JDK-Version definiert *(siehe Listing 7)*. SDKMAN! kann diese Datei auch mit aktuell genutzten Versionen mit dem Befehl `sdk env init` anlegen.

Wenn die Autodetection von SDKMAN! Eingeschaltet ist, dann wechselt SDKMAN! Automatisch auf die richtige Version, beziehungsweise schlägt vor, sie zu installieren *(siehe Listing 8)*. Die Entwickler können die Installation aber auch manuell mit `sdk env install` anstoßen.

Ähnliche Werkzeuge gibt es auch für andere Ökosysteme. *nvm* *[22]* verwaltet beispielsweise Node-Versionen, *asdf* *[23]* verwaltet Werkzeuge aus verschiedenen Ökosystemen.

## Arbeiten mit Dateien

Wenn eine Entwicklerin schnell auf den Inhalt einer Datei zugreifen möchte, dann wird gerne auf cat (ist oft in der Standardinstallation eines Systems dabei) *[24]* verwiesen *(siehe Listing 9)*.

`cat` ist super geeignet, wenn der Inhalt einer Datei mithilfe von Pipes *[25]* mit anderen Werkzeugen weiterverarbeitet werden soll oder mehrere Dateien zusammengeführt werden sollen.

`cat` ist nicht sehr hilfreich, wenn man den Inhalt nur anschauen möchte und dafür Syntaxhervorhebung und Zeilenangaben braucht. Hier hilft das Werkzeug bat *[26]* weiter. Es ist in der Benutzung

```
→ cat pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.2</version>
    </parent>

    <groupId>com.github.sparsick</groupId>
    <artifactId>spring-boot-example</artifactId>
    <version>1.5.0</version>
    <name>spring-boot-example</name>
    <description>Demo project for Spring Boot</description>
```

*Listing 9: Beispiel mit cat*



```
File: pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
>
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.2</version>
    </parent>

    <groupId>com.github.sparsick</groupId>
    <artifactId>spring-boot-example</artifactId>
    <version>1.5.0</version>
    <name>spring-boot-example</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>17</java.version>
        <selenium.version>4.11.0</selenium.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
```

*Abbildung 1: Dateien-Anzeige mit bat (© bat, Screenshot von Sandra Parsick)*

```
spring-boot-demo on  v24.0.7 …
→ ag hero
README.md
12:Application URL: `http://localhost:8080/hero`

application-monitoring/grafana-config/dashboards/hero-demo.json
60:          "expr": "http_server_requests_seconds_max{uri=\"/hero\"}",
68:      "title": "Hero Search Request Time",
111:         "expr": "rate(hero_usage_add_total[60s])",
115:         "legendFormat": "Add Hero Feature",
119:         "expr": "rate(hero_usage_search_total[60s])",
122:         "legendFormat": "Search Hero Feature",
130:     "title": "Hero Feature",
206:         "expr": "http_server_requests_seconds_max{uri=\"/hero\",status='200'}",
209:         "legendFormat": "Hero List",
226:     "title": "Hero Request Time",
300:         "expr": "rate(http_server_requests_seconds_count{uri=\"/hero/list\", status=\"200\"}[5s])",
303:         "legendFormat": "Hero List",
307:         "expr": "rate(http_server_requests_seconds_count{uri=\"/hero/new\", method=\"GET\"}[5s])",
310:         "legendFormat": "Add new Hero",
314:         "expr": "rate(http_server_requests_seconds_count{uri=\"/hero\", method=\"GET\"}[5s])",
317:         "legendFormat": "Hero Search View",
325:     "title": "Hero Request per Time",
399:         "expr": "hero_repository_experiment_all_hero_runs_total{match = \"no\"}",
406:         "expr": "hero_repository_experiment_all_hero_runs_total{match = \"yes\"}",
413:         "expr": "sum(hero_repository_experiment_all_hero_runs_total)",
424:     "title": "Hero Repository All Hero Experiment",
500:  "title": "Hero Demo",

src/test/java/com/github/sparsick/springbootexample/hero/HeroApplicationTests.java
1:package com.github.sparsick.springbootexample.hero;
8:public class HeroApplicationTests {
```

*Abbildung 2: Suchergebnisse mit ag unter der Verwendung von Silversearcher (© ag, Screenshot von Sandra Parsick)*

leichtgewichtig wie `cat`, bietet aber Syntaxhervorhebung und Zeilenangaben an, bei Wunsch zeigt es auch Git-Änderung pro Zeile an *(siehe Abbildung 1)*.

Wenn der Entwickler eine Menge an Dateien durchsuchen und dafür gerne die Shell verwenden möchte, wird er oftmals auf `find` *[27]* und `grep` *[28]* verwiesen. Das sind mächtige Werkzeuge, aber nicht intuitiv zu bedienen und es fehlen bei den Ergebnissen Kontextinformationen, die für einen Entwickler interessant sind. Das Werkzeug *Silversearcher* *[29]* liefert genau diese Funktionalität.

In der Standardbenutzung ag `suchbegriff` *(siehe Abbildung 2)* listet *Silversearcher* alle Stellen inklusive Dateipfad und Zeilenangaben auf, in denen der gesuchte Begriff vorkommt.

```
spring-boot-demo on  master on   v24.0.7 …
→ ag --json hero
application-monitoring/grafana-config/dashboards/hero-demo.json
60:          "expr": "http_server_requests_seconds_max{uri=\"/hero\"}",
68:      "title": "Hero Search Request Time",
111:          "expr": "rate(hero_usage_add_total[60s])",
115:          "legendFormat": "Add Hero Feature",
119:          "expr": "rate(hero_usage_search_total[60s])",
122:          "legendFormat": "Search Hero Feature",
130:      "title": "Hero Feature",
206:          "expr": "http_server_requests_seconds_max{uri=\"/hero\",status='200'}",
209:          "legendFormat": "Hero List",
226:      "title": "Hero Request Time",
300:          "expr": "rate(http_server_requests_seconds_count{uri=\"/hero/list\", status=\"200\"}[5s])",
303:          "legendFormat": "Hero List",
307:          "expr": "rate(http_server_requests_seconds_count{uri=\"/hero/new\", method=\"GET\"}[5s])",
310:          "legendFormat": "Add new Hero",
314:          "expr": "rate(http_server_requests_seconds_count{uri=\"/hero\", method=\"GET\"}[5s])",
317:          "legendFormat": "Hero Search View",
325:      "title": "Hero Request per Time",
399:          "expr": "hero_repository_experiment_all_hero_runs_total{match = \"no\"}",
406:          "expr": "hero_repository_experiment_all_hero_runs_total{match = \"yes\"}",
413:          "expr": "sum(hero_repository_experiment_all_hero_runs_total)",
424:      "title": "Hero Repository All Hero Experiment",
500:  "title": "Hero Demo",
```

*Abbildung 3: Gefilterte Suchergebnisse mit ag (© ag, Screenshot von Sandra Parsick)*

Möchte eine Entwicklerin die Suche auf bestimmte Dateitypen einschränken, kann sie es über die Option `--datentyp` (zum Beispiel `--json` für eine Einschränkung auf JSON-Dateien, *siehe Abbildung 3*). Die Option `--list-file-types` listet alle unterstützen Datentypen auf.

Braucht ein Entwickler doch nur eine Auflistung aller Dateien, in der ein Suchbegriff auftaucht, kann er die Ausgabe auf diese Information mit der Option `--files-with-matches` einschränken *(siehe Abbildung 4)*.

*Silversearcher* bietet noch weitere Optionen an, um die Suche und Ausgabe auf die eigenen Bedürfnisse anzupassen. Besonders wenn Entwickler JSON oder YAML genauer durchsuchen wollen oder einfach nur Kontext-basiert parsen wollen, stößt auch *Silversearcher* an seine Grenzen. Hier bietet es sich an, darauf spezialisierte Werkzeuge zu benutzen, wie zum Beispiel *jq* (für JSON) *[30]* oder *yq* (für YAML) *[31]*. Es sind zwei verschiedene Werkzeuge, deren Benutzung jedoch ähnlich gehalten ist.

In dem folgenden Beispiel nehmen wir an, dass Entwickler eine JSON-Datei *(siehe Listing 10)* durchsuchen möchten.

Das erste Problem ist, dass die Datei nicht formatiert ist und somit für eine Entwicklerin schwer zu lesen ist. Mit `` `cat starships.json | jq.` `` lässt sich die Datei formatieren *(siehe Listing 11)*.

Möchte der Entwickler aus dem JSON nur die Werte, die unter dem Schlüssel `results` liegen, gibt er `cat starships.json| jq .results` ein. Möchte er es weiter einschränken, zum Beispiel nur die Namen der Sternenschiffe innerhalb des Arrays, gibt er `cat starships.json| jq '.results.[].name'` ein.

```
spring-boot-demo on  master …
→ ag --files-with-matches hero
README.md
application-monitoring/grafana-config/dashboards/hero-demo.json
src/test/java/com/github/sparsick/springbootexample/hero/HeroApplicationTests.java
src/test/java/com/github/sparsick/springbootexample/hero/HeroStartPageIT.java
src/main/java/com/github/sparsick/springbootexample/hero/HeroConfiguration.java
src/main/java/com/github/sparsick/springbootexample/hero/universum/NewHeroModel.java
src/main/java/com/github/sparsick/springbootexample/hero/universum/HeroController.java
src/main/java/com/github/sparsick/springbootexample/hero/universum/Hero.java
src/main/java/com/github/sparsick/springbootexample/hero/HeroApplication.java
src/main/java/com/github/sparsick/springbootexample/hero/universum/MongoDbHeroRepository.java
src/main/resources/templates/hero/hero.list.html
src/main/java/com/github/sparsick/springbootexample/hero/universum/HeroRepository.java
src/main/resources/templates/hero/hero.search.html
src/main/java/com/github/sparsick/springbootexample/hero/universum/ComicUniversum.java
src/main/resources/templates/hero/hero.new.html
src/main/java/com/github/sparsick/springbootexample/hero/universum/HeroMongoRepository.java
src/main/java/com/github/sparsick/springbootexample/hero/universum/EmbbededHeroRepository.java
```

*Abbildung 4: Auflistung der Dateinamen (© ag, Screenshot von Sandra Parsick)*



```
{"count":36,"next":"https://swapi.dev/api/st
arships/?page=2","previous":null,"results":
[{"name":"CR90 corvette","model":"CR90 corv
ette","manufacturer":"Corellian Engineering
Corporation","cost_in_credits":"3500000","length"
:"150","max_atmosphering_speed":"950","crew":"30-
165","passengers":"600","cargo_capacity":"3000000",
"consumables":"1 year","hyperdrive_rating":"2.0","M
GLT":"60","starship_class":"corvette","pilots":[],"
films":["https://swapi.dev/api/films/1/","https://
swapi.dev/api/films/3/","https://swapi.dev/api/
films/6/"],"created":"2014-12-10T14:20:33.369000Z","e
dited":"2014-12-20T21:23:49.867000Z","url":"https://
swapi.dev/api/starships/2/"}]}%
```

*Listing 10: Unformatierte, schlecht lesbare JSON-Datei (Ausschnitt)*

Das Werkzeug yq ist in der Benutzung ähnlich, nur dass es sich auf YAML-Dateien spezialisiert hat.

## HTTP-Schnittstellen aufrufen

Wenn es darum geht HTTP-Schnittstellen aufzurufen, dann wird gerne auf `curl` oder `wget` *[32]* verwiesen. Diese Werkzeuge sind sehr mächtig, jedoch nicht sehr intuitiv zu bedienen. Oft braucht es etwas leichtgewichtigeres, um zum Beispiel ein REST-API zu testen. Für diesen Anwendungsfall gibt es *httpie [33]*. Es besitzt eine intuitive Schnittstelle und liefert alle wichtigen Informationen auf einen Blick *(siehe Listing 12)*.

Der Aufruf erfolgt nach dem Muster `http HTTP-METHOD url`. Möchte eine Entwicklerin nicht alle Informationen, kann sie diese über Optionen (`--headers`, `--meta`, `--body`, *siehe Listing 13*) einschränken.

Über weitere Optionen kann man auch die SSL- und Authentisierungseinstellungen steuern.

## Tipps und Tricks

Die letzten Abschnitte haben einen kleinen Einblick gegeben, wie die Shell bei alltäglichen Entwickleraufgaben helfen kann. Doch gerade Anfänger sind mit vielen Sachen, die in der Shell passieren können, etwas überfordert. Daher nachfolgend ein paar Tipps, die das Arbeiten auf der Shell vereinfachen.

1. **Frag deine Kollegen:** Schaue beim Pair-Programming darauf, wie deine Kollegen bestimmte Aufgaben auf der Shell (vielleicht nicht nur dort) lösen und frage nach, was sie gerade gemacht haben.
2. **Benutze Cheat Sheets:** Gerade bei mächtigen Werkzeugen verliert man schnell den Überblick, was alles möglich ist. Cheat Sheets helfen dabei, den Überblick zu behalten.
3. **Benutze Man Pages oder die –help-Option:** Zu jedem Werkzeug gibt es die Hilfsoption oder eine Man Page (`man werkzeug`), die die Benutzung des Werkzeuges und ihre Optionen erklärt.

Es gibt noch weitere Quellen, die gut erklären, wie Befehle auf der Shell funktionieren: Die Webseite *Explain Shell [34]* erzeugt zum Beispiel eine genaue Erklärung für jeden Befehl, den man dort eingibt *(siehe Abbildung 5)*.

Die Webseite *tldr pages [35]* erklärt Befehle über die Man Page hinaus anhand von Beispielen.

```
{
  "count": 36,
  "next": "https://swapi.dev/api/starships/?page=2",
  "previous": null,
  "results": [
     {
        "name": "CR90 corvette",
        "model": "CR90 corvette",
        "manufacturer": "Corellian Engineering Corporation",
        "cost_in_credits": "3500000",
        "length": "150",
        "max_atmosphering_speed": "950",
        "crew": "30-165",
        "passengers": "600",
        "cargo_capacity": "3000000",
        "consumables": "1 year",
        "hyperdrive_rating": "2.0",
        "MGLT": "60",
        "starship_class": "corvette",
        "pilots": [],
        "films": [
           "https://swapi.dev/api/films/1/",
           "https://swapi.dev/api/films/3/",
           "https://swapi.dev/api/films/6/"
        ],
        "created": "2014-12-10T14:20:33.369000Z",
        "edited": "2014-12-20T21:23:49.867000Z",
        "url": "https://swapi.dev/api/starships/2/"
     }
  ]
}
```


*Listing 11: Beispiel für eine formatierte JSON-Datei (Ausschnitt)*

```
→ http GET https://swapi.dev/api/starships/9/
HTTP/1.1 200 OK
Allow: GET, HEAD, OPTIONS
Connection: keep-alive
Content-Type: application/json
Date: Fri, 12 Jan 2024 10:08:02 GMT
ETag: "058c95fce38484128f1c3f2e5dd04d50"
Server: nginx/1.16.1
Strict-Transport-Security: max-age=15768000
Transfer-Encoding: chunked
Vary: Accept, Cookie
X-Frame-Options: SAMEORIGIN

{
     "MGLT": "10",
     "cargo_capacity": "1000000000000",
     "consumables": "3 years",
     "cost_in_credits": "1000000000000",
     "created": "2014-12-10T16:36:50.509000Z",
     "crew": "342,953",
     "edited": "2014-12-20T21:26:24.783000Z",
     "films": [
           "https://swapi.dev/api/films/1/"
     ],
     "hyperdrive_rating": "4.0",
     "length": "120000",
     "manufacturer": "Imperial Department of Military Research, Sienar Fleet Systems",
     "max_atmosphering_speed": "n/a",
     "model": "DS-1 Orbital Battle Station",
     "name": "Death Star",
     "passengers": "843,342",
     "pilots": [],
     "starship_class": "Deep Space Mobile Battlestation",
     "url": "https://swapi.dev/api/starships/9/"
}
```

*Listing 12: Beispiel httpie*

## Quellen


[1] Wikipedia zu Terminal (Computer) Webseite: *https://de.wikipedia.org/wiki/Terminal_(Computer)*

[2] Terminal Konsole Webseite: *https://konsole.kde.org/*

[3] Terminal GNOME Terminal Webseite: *https://wiki.gnome.org/Apps/Terminal*

[4] Terminal Tilix Webseite: *https://gnunn1.github.io/tilix-web/*

[5] Terminal Terminal für den Mac Webseite: *https://support.apple.com/de-de/guide/terminal/welcome/mac*

[6] Terminal iterm2 Webseite: *https://iterm2.com/index.html*

[7] Terminal Windows Terminal Webseite: *https://learn.microsoft.com/de-de/windows/terminal/*

[8] Terminal fig Webseite: *https://fig.io/*

[9] Terminal warp Webseite: *https://www.warp.dev/*

[10] Terminal hyper Webseite: *https://hyper.is/*

[11] Wikipedia zu Shell (Betriebssystem) Webseite: *https://de.wikipedia.org/wiki/Shell_(Betriebssystem)*

[12] Bash Webseite: *https://www.gnu.org/software/bash/*

[13] Zsh Webseite: *https://www.zsh.org/*

[14] fish Shell Webseite: *https://fishshell.com/*

[15] Oh My Zsh Webseite: *https://ohmyz.sh/*

[16] Oh My Bash Webseite: *https://ohmybash.nntoan.com/*

[17] Oh My Posh Webseite: *https://ohmyposh.dev*

[18] cUrl Webseite: *https://curl.se/*

[19] Git Webseite: *https://git-scm.com/*

[20] Theme Starship Webseite: *https://starship.rs/*

[21] SDKMAN! Webseite: *https://sdkman.io/*

[22] nvm Webseite: *https://github.com/nvm-sh/nvm*

[23] asdf Webseite: *https://asdf-vm.com/*

[24] cat Webseite: *https://wiki.ubuntuusers.de/cat/*

[25] Pipes / Umleitungen Webseite: *https://wiki.ubuntuusers.de/Shell/Umleitungen/*

[26] bat Webseite: *https://github.com/sharkdp/bat*

[27] find Webseite: *https://wiki.ubuntuusers.de/find/*

[28] grep Webseite: *https://wiki.ubuntuusers.de/grep/*

[29] The Silver Searcher Webseite: *https://geoff.greer.fm/ag/*

[30] jq Webseite: *https://jqlang.github.io/jq/*

[31] yq Webseite: *https://mikefarah.gitbook.io/yq/*

[32] wget Webseite: *https://www.gnu.org/software/wget/*

[33] httpie Webseite: *https://httpie.io/*

[34] Webseite: *https://explainshell.com/*

[35] Webseite: *https://tldr.sh/*


```
→ http GET https://swapi.dev/api/starships/9/ --headers
HTTP/1.1 200 OK
Allow: GET, HEAD, OPTIONS
Connection: keep-alive
Content-Type: application/json
Date: Fri, 12 Jan 2024 10:18:34 GMT
ETag: "058c95fce38484128f1c3f2e5dd04d50"
Server: nginx/1.16.1
Strict-Transport-Security: max-age=15768000
Transfer-Encoding: chunked
Vary: Accept, Cookie
X-Frame-Options: SAMEORIGIN
```

*Listing 13: Beispiel httpie mit Optionen*


**Sandra Parsick**

Sandra Parsick ist Java Champion und arbeitet als freiberufliche Softwareentwicklerin und Consultant im Java-Umfeld. Seit 2008 beschäftigt sie sich mit agiler Softwareentwicklung in verschiedenen Rollen. Ihre Schwerpunkte liegen im Bereich Java Enterprise, Cloud, Software Craftsmanship und in der Automatisierung von Entwicklungsprozessen. Darüber schreibt sie gerne Artikel und spricht auf Konferenzen. In ihrer Freizeit engagiert sie sich in verschiedenen Programmkomitees und Community-Gruppen.



*Abbildung 5: Beispielausgabe in explainshell.com (© explainshell.com, Screenshot von Sandra Parsick)*


## Mitglieder des iJUG

01 BED-Con e.V.
02 Clojure User Group Düsseldorf
03 DOAG e.V.
04 EuregJUG Maas-Rhine
05 JUG Augsburg
06 JUG Berlin-Brandenburg
07 JUG Bremen
08 JUG Bielefeld
09 JUG Bonn
10 JUG Darmstadt
11 JUG Deutschland e.V.
12 JUG Dortmund
13 JUG Düsseldorf rheinjug
14 JUG Erlangen-Nürnberg
15 JUG Freiburg
16 JUG Goldstadt
17 JUG Görlitz
18 JUG Hannover
19 JUG Hessen
20 JUG HH
21 JUG Ingolstadt e.V.
22 JUG Kaiserslautern
23 JUG Karlsruhe
24 JUG Köln
25 Kotlin User Group Düsseldorf
26 JUG Mainz
27 JUG Mannheim
28 JUG München
29 JUG Münster
30 JUG Oberland
31 JUG Ostfalen
32 JUG Paderborn
33 JUG Saxony
34 JUG Stuttgart e.V.
35 JUG Switzerland
36 JSUG
37 Lightweight JUG München
38 SUG Deutschland e.V.
39 JUG Thüringen
40 JUG Saarland
41 JUG Duisburg
42 JUG Frankfurt

iJUG Verbund
www.ijug.eu

## Impressum

Java aktuell wird vom Interessenverbund der Java User Groups e.V. (iJUG) (Tempelhofer Weg 64, 12347 Berlin, *www.ijug.eu*) herausgegeben. Es ist das User-Magazin rund um die Programmiersprache Java im Raum Deutschland, Österreich und Schweiz. Es ist unabhängig von Oracle und vertritt weder direkt noch indirekt deren wirtschaftliche Interessen. Vielmehr vertritt es die Interessen der Anwender an den Themen rund um die Java-Produkte, fördert den Wissensaustausch zwischen den Lesern und informiert über neue Produkte und Technologien.

Java aktuell wird verlegt von der DOAG Dienstleistungen GmbH, Tempelhofer Weg 64, 12347 Berlin, Deutschland, gesetzlich vertreten durch den Geschäftsführer Fried Saacke, deren Unternehmensgegenstand Vereinsmanagement, Veranstaltungsorganisation und Publishing ist.

DOAG e.V. hält 100 Prozent der Stammeinlage der DOAG Dienstleistungen GmbH. DOAG e.V. wird gesetzlich durch den Vorstand vertreten; Vorsitzender: Björn Bröhl. DOAG e.V. informiert kompetent über alle Oracle-Themen, setzt sich für die Interessen der Mitglieder ein und führen einen konstruktiv-kritischen Dialog mit Oracle.

Redaktion:
Sitz: DOAG Dienstleistungen GmbH
ViSdP: Fried Saacke
Redaktionsleitung: Lisa Damerow
Kontakt: redaktion@ijug.eu

Redaktionsbeirat:
Andreas Badelt, Marcus Fihlon, Markus Karg, Manuel Mauky, Bernd Müller, Benjamin Nothdurft, Daniel van Ross, Bennet Schulz

Titel, Gestaltung und Satz:
Alexander Kermas,
DOAG Dienstleistungen GmbH

Bildnachweis:
Titel: Bild © Designed by catalyststuff
*https://freepik.com*
S. 3 + 4: Bild © ant
*https://stock.adobe.com*

Anzeigen:
DOAG Dienstleistungen GmbH
Kontakt: sponsoring@doag.org
Mediadaten und Preise:
*www.doag.org/go/mediadaten*

Druck:
WIRmachenDRUCK GmbH
*www.wir-machen-druck.de*



## Inserentenverzeichnis