2.4 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	2 Sich auf einem Linux-System zurechtfinden
Lernziel:	2.4 Erstellen, Verschieben und Löschen von Dateien
Lektion:	1 von 1

Einführung

Diese Lektion behandelt die Verwaltung von Dateien und Verzeichnissen unter Linux mit Hilfe von Befehlszeilenwerkzeugen.

Eine Datei ist eine Sammlung von Daten mit einem Namen und einer Reihe von Attributen. Wenn Sie beispielsweise Fotos von Ihrem Handy auf einen Computer übertragen und ihnen beschreibende Namen geben, haben Sie eine Reihe von Bilddateien auf Ihrem Computer. Diese Dateien haben Attribute wie die Zeit des letzten Zugriffs auf die Datei oder den Zeitpunkt der letzten Änderung.

Ein Verzeichnis ist eine spezielle Art von Datei zum Organisieren von Dateien. Stellen Sie sich Verzeichnisse als Aktenordner vor, in denen man Dokumente aufbewahrt. Aber anders als bei Aktenordnern aus Pappe können Sie in einem Verzeichnis weitere Verzeichnisse ablegen.

Die Befehlszeile ist der effektivste Weg, um Dateien auf einem Linux-System zu verwalten. Die Shell- und Kommandozeilen-Tools verfügen über Funktionen, die die Arbeit auf der Kommandozeile schneller und einfacher machen als ein grafischer Dateimanager.

In diesem Abschnitt verwenden Sie die Befehle `ls`, `mv`, `cp`, `pwd`, `find`, `touch`, `rm`, `rmdir`, `echo`, `cat` und `mkdir` zur Verwaltung und Organisation von Dateien und Verzeichnissen.

Groß-/Kleinschreibung beachten

Im Gegensatz zu Microsoft Windows ist auf Linux-Systemen bei Datei- und Verzeichnisnamen Groß-/Kleinschreibung zu unterscheiden, d.h. `/etc/` und `/ETC/` bezeichnen unterschiedliche Verzeichnisse:

```
$ cd /
$ ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
$ cd ETC
bash: cd: ETC: No such file or directory
$ pwd
/
$ cd etc
$ pwd
/etc
```

Das `pwd` zeigt Ihnen das Verzeichnis, in dem Sie sich gerade befinden. Wie Sie sehen, hat der Wechsel zu `/ETC` nicht funktioniert, da es kein solches Verzeichnis gibt. Der Wechsel in das existierende Verzeichnis `/etc` ist hingegen gelungen.

Verzeichnisse erstellen

Der Befehl `mkdir` wird verwendet, um Verzeichnisse zu erstellen.

Wir wollen nun ein neues Verzeichnis in unserem Heimatverzeichnis erstellen:

```
$ cd ~
$ pwd
/home/user
$ ls
Desktop  Documents  Downloads
$ mkdir linux_essentials-2.4
$ ls
Desktop  Documents  Downloads  linux_essentials-2.4
$ cd linux_essentials-2.4
$ pwd
/home/emma/linux_essentials-2.4
```

In dieser Lektion führen wir sämtliche Befehle in diesem Verzeichnis oder in einem seiner Unterverzeichnisse aus.

Um von jeder anderen Position in Ihrem Dateisystem einfach in das Lektionsverzeichnis zurückzukehren, können Sie den folgenden Befehl verwenden:

```
$ cd ~/linux_essentials-2.4
```

Die Shell interpretiert das Zeichen `~` als Ihr Home-Verzeichnis.

Erstellen Sie im Lektionsverzeichnis weitere Verzeichnisse, die wir für die Übungen verwenden werden. Setzen Sie alle Verzeichnisnamen, getrennt durch Leerzeichen, hinter `mkdir`:

```
$ mkdir creating moving copying/files copying/directories
deleting/directories deleting/files globs
mkdir: cannot create directory ‘copying/files’: No such file or directory
mkdir: cannot create directory ‘copying/directories’: No such file or
directory
mkdir: cannot create directory ‘deleting/directories’: No such file or
directory
mkdir: cannot create directory ‘deleting/files’: No such file or directory
$ ls
creating  globs  moving
```

Beachten Sie die Fehlermeldung, dass nur `moving`, `globs` und `creating` erstellt wurden. Die Verzeichnisse `copying` und `deleting` existieren noch nicht. `mkdir` erstellt standardmäßig kein Verzeichnis innerhalb eines Verzeichnisses, das nicht bereits existiert. Die Option `-p` oder `--parents` weist `mkdir` an, übergeordnete Verzeichnisse zu erstellen, wenn sie nicht vorhanden sind. Probieren Sie den gleichen Befehl `mkdir` mit der Option `-p`:

```
$ mkdir -p creating moving copying/files copying/directories
deleting/directories deleting/files globs
```

Jetzt bekommen Sie keine Fehlermeldungen mehr. Lassen Sie uns sehen, welche Verzeichnisse jetzt existieren:

```
$ find
.
./creating
./moving
./globs
```

```
./copying
./copying/files
./copying/directories
./deleting
./deleting/directories
./deleting/files
```

Das Programm `find` wird normalerweise verwendet, um nach Dateien und Verzeichnissen zu suchen, aber ohne Optionen zeigt es eine Liste aller Dateien, Verzeichnisse und Unterverzeichnisse des aktuellen Verzeichnisses.

Tip Wenn Sie den Inhalt eines Verzeichnisses mit `ls` auflisten, sind die Optionen `-t` und `-r` besonders praktisch: Sie sortieren die Ausgabe nach Zeit (`-t`) und kehren die Sortierreihenfolge um (`-r`); in diesem Fall stehen die neuesten Dateien am Ende der Ausgabe.

Dateien erstellen

Normalerweise werden Dateien von den Programmen erstellt, die mit den in den Dateien gespeicherten Daten arbeiten. Eine leere Datei erstellen Sie mit dem Befehl `touch`. Wenn Sie `touch` auf einer bestehenden Datei ausführen, ändert sich der Inhalt der Datei nicht, aber der Zeitstempel der Dateiänderung wird aktualisiert.

Führen Sie den folgenden Befehl aus, um einige Dateien für die Lektion zum Thema “Globbing” zu erstellen:

```
$ touch globs/question1 globs/question2012 globs/question23
globs/question13 globs/question14
$ touch globs/star10 globs/star1100 globs/star2002 globs/star2013
```

Lassen Sie uns überprüfen, ob alle Dateien im Verzeichnis `globs` vorhanden sind:

```
$ cd globs
$ ls
question1   question14    question23  star1100  star2013
question13  question2012  star10      star2002
```

`touch` hat also die Dateien erstellt. Sie können den Inhalt einer Textdatei mit dem Befehl `cat` ansehen und ihn an einer der Dateien ausprobieren, die Sie gerade erstellt haben:

```
$ cat question14
```

Da `touch` leere Dateien erzeugt, sollten Sie keine Ausgabe erhalten. Sie können `echo` mit `>` verwenden, um einfache Textdateien zu erstellen:

```
$ echo hello > question15
$ cat question15
hello
```

`echo` zeigt Text auf der Kommandozeile an. Das Zeichen `>` weist die Shell an, die Ausgabe eines Befehls in die angegebene Datei (statt ins Terminal) zu schreiben, was dazu führt, dass die Ausgabe von `echo`, in diesem Fall `hello`, in die Datei `question15` geschrieben wird. Das ist nicht spezifisch für `echo`, das geht mit jedem anderen Befehl.

Umbenennen von Dateien

Dateien werden mit dem Befehl `mv` verschoben und umbenannt.

Stellen Sie Ihr Arbeitsverzeichnis auf das Verzeichnis `moving` ein:

```
$ cd ~/linux_essentials-2.4/moving
```

Erstellen Sie einige Dateien zum Üben. Mit den notwendigen Befehlen sollten Sie bereits vertraut sein:

```
$ touch file1 file22
$ echo file3 > file3
$ echo file4 > file4
$ ls
file1  file22  file3  file4
```

Angenommen `file22` ist ein Tippfehler und sollte `file2` heißen. Korrigieren Sie es mit dem Befehl `mv`. Beim Umbenennen einer Datei ist das erste Argument der aktuelle Name, das zweite der neue Name:

```
$ mv file22 file2
$ ls
file1  file2  file3  file4
```

Seien Sie vorsichtig mit dem Befehl `mv`: Wenn Sie eine Datei in eine bereits bestehende Datei umbenennen, wird diese überschrieben. Testen wir dies mit `file3` und `file4`:

```
$ cat file3
file3
$ cat file4
file4
$ mv file4 file3
$ cat file3
file4
$ ls
file1  file2  file3
```

Beachten Sie, dass der Inhalt von `file3` jetzt `file4` ist. Verwenden Sie die Option `-i`, um `mv` anzuweisen, vor dem Überschreiben einer bestehenden Datei eine Bestätigung anzufordern:

```
$ touch file4 file5
$ mv -i file4 file3
mv: overwrite ‘file3’? y
```

Verschieben von Dateien

Dateien werden mit dem Befehl `mv` von einem Verzeichnis in ein anderes verschoben.

Erstellen Sie ein paar Verzeichnisse, in die Sie Dateien verschieben können:

```
$ cd ~/linux_essentials-2.4/moving
$ mkdir dir1 dir2
$ ls
dir1  dir2  file1  file2  file3  file5
```

Verschieben Sie `file1` in `dir1`:

```
$ mv file1 dir1
$ ls
dir1  dir2  file2  file3  file5
$ ls dir1
file1
```

Beachten Sie, dass das letzte Argument von `mv` das Zielverzeichnis ist. Wenn das letzte Argument von `mv` ein Verzeichnis ist, werden Dateien in dieses Verzeichnis verschoben, wobei mehrere Dateien in einem einzigen `mv` Befehl angegeben werden können:

```
$ mv file2 file3 dir2
$ ls
dir1  dir2  file5
$ ls dir2
file2  file3
```

Es ist auch möglich, mit `mv` Verzeichnisse zu verschieben und umzubenennen. Benennen Sie `dir1` in `dir3` um:

```
$ ls
dir1  dir2  file5
$ ls dir1
file1
$ mv dir1 dir3
$ ls
dir2  dir3  file5
$ ls dir3
file1
```

Löschen von Dateien und Verzeichnissen

Der Befehl `rm` löscht Dateien und Verzeichnisse, während der Befehl `rmdir` nur Verzeichnisse löschen kann. Räumen wir das Verzeichnis `moving` durch Löschen von `file5` auf:

```
$ cd ~/linux_essentials-2.4/moving
$ ls
dir2  dir3  file5
$ rmdir file5
rmdir: failed to remove ‘file5’: Not a directory
$ rm file5
$ ls
dir2  dir3
```

Standardmäßig kann `rmdir` nur leere Verzeichnisse löschen, darum mussten wir `rm` verwenden, um eine normale Datei zu löschen. Versuchen Sie, das Verzeichnis `deleting` zu löschen:

```
$ cd ~/linux_essentials-2.4/
$ ls
copying  creating  deleting  globs  moving
$ rmdir deleting
rmdir: failed to remove ‘deleting’: Directory not empty
$ ls -l deleting
total 0
drwxrwxr-x. 2 emma emma 6 Mar 26 14:58 directories
drwxrwxr-x. 2 emma emma 6 Mar 26 14:58 files
```

Standardmäßig weigert sich `rmdir`, ein Verzeichnis zu löschen, das nicht leer ist. Verwenden Sie `rmdir`, um eines der leeren Unterverzeichnisse im Verzeichnis `deleting` zu entfernen:

```
$ ls -a deleting/files
.  ..
$ rmdir deleting/files
$ ls -l deleting
directories
```

Das Löschen einer großen Anzahl von Dateien oder tiefer Verzeichnisstrukturen mit vielen Unterverzeichnissen mag aufwändig scheinen, ist aber eigentlich einfach. `rm` funktioniert standardmäßig nur bei normalen Dateien. Die Option `-r` wird verwendet, um dieses Verhalten zu überschreiben. Aber Vorsicht, `rm -r` ist brandgefährlich! Mit der Option `-r` löscht `rm` nicht nur alle Verzeichnisse, sondern alles in diesem Verzeichnis, einschließlich der Unterverzeichnisse und deren Inhalte. Sehen Sie selbst, wie `rm -r` funktioniert:

```
$ ls
copying  creating  deleting  globs  moving
$ rm deleting
rm: cannot remove ‘deleting’: Is a directory
$ ls -l deleting
total 0
drwxrwxr-x. 2 emma emma 6 Mar 26 14:58 directories
$ rm -r deleting
$ ls
copying  creating  globs  moving
```

Beachten Sie, wie `deleting` verschwunden ist, obwohl es nicht leer war. Wie `mv` hat `rm` eine `-i` Option, um vor der Ausführung eine Bestätigung anzufordern. Nutzen Sie `rm -ri`, um Verzeichnisse aus `moving` zu entfernen, die nicht mehr benötigt werden:

```
$ find
.
./creating
./moving
./moving/dir2
./moving/dir2/file2
./moving/dir2/file3
./moving/dir3
./moving/dir3/file1
./globs
./globs/question1
./globs/question2012
./globs/question23
./globs/question13
./globs/question14
./globs/star10
./globs/star1100
./globs/star2002
./globs/star2013
./globs/question15
./copying
./copying/files
./copying/directories
$ rm -ri moving
rm: descend into directory ‘moving’? y
```

```
rm: descend into directory ‘moving/dir2’? y
rm: remove regular empty file ‘moving/dir2/file2’? y
rm: remove regular empty file ‘moving/dir2/file3’? y
rm: remove directory ‘moving/dir2’? y
rm: descend into directory ‘moving/dir3’? y
rm: remove regular empty file ‘moving/dir3/file1’? y
rm: remove directory ‘moving/dir3’? y
rm: remove directory ‘moving’? y
```

Kopieren von Dateien und Verzeichnissen

Mit dem Befehl `cp` werden Dateien und Verzeichnisse kopiert. Kopieren Sie einige Dateien in das Verzeichnis `copying`:

```
$ cd ~/linux_essentials-2.4/copying
$ ls
directories  files
$ cp /etc/nsswitch.conf files/nsswitch.conf
$ cp /etc/issue /etc/hostname files
```

Wenn das letzte Argument ein Verzeichnis ist, erstellt `cp` eine Kopie der vorherigen Argumente innerhalb des Verzeichnisses. Wie bei `mv` kann man mehrere Dateien auf einmal angeben, solange das Ziel ein Verzeichnis ist.

Wenn beide Operanden von `cp` Dateien sind und beide Dateien existieren, überschreibt `cp` die zweite Datei mit einer Kopie der ersten Datei. Wir wollen das üben, indem wir die Datei `issue` mit der Datei `hostname` überschreiben:

```
$ cd ~/linux_essentials-2.4/copying/files
$ ls
hostname  issue  nsswitch.conf
$ cat hostname
mycomputer
$ cat issue
Debian GNU/Linux 9 \n \l

$ cp hostname issue
$ cat issue
mycomputer
```

Versuchen wir nun, eine Kopie des Verzeichnisses `files` im Verzeichnis `directories` zu erstellen:

```
$ cd ~/linux_essentials-2.4/copying
$ cp files directories
cp: omitting directory ‘files’
```

Wie Sie sehen, funktioniert `cp` standardmäßig nur bei einzelnen Dateien. Um ein Verzeichnis zu kopieren, verwenden Sie die Option `-r`. Beachten Sie, dass die Option `-r` bewirkt, dass `cp` den Inhalt des Verzeichnisses, das Sie kopieren, ebenfalls kopiert:

```
$ cp -r files directories
$ find
.
./files
./files/nsswitch.conf
./files/fstab
./files/hostname
./directories
```

```
./directories/files
./directories/files/nsswitch.conf
./directories/files/fstab
./directories/files/hostname
```

Sehen Sie, dass, wenn ein bestehendes Verzeichnis als Ziel verwendet wurde, `cp` eine Kopie des Quellverzeichnisses innerhalb des Verzeichnisses erstellt? Wenn das Ziel nicht existiert, wird es erstellt und mit dem Inhalt des Quellverzeichnisses gefüllt:

```
$ cp -r files files2
$ find
.
./files
./files/nsswitch.conf
./files/fstab
./files/hostname
./directories
./directories/files
./directories/files/nsswitch.conf
./directories/files/fstab
./directories/files/hostname
./files2
./files2/nsswitch.conf
./files2/fstab
./files2/hostname
```

Globbing

Was allgemein als *Globbing* bezeichnet wird, ist eine einfache Musterabgleichssprache. Kommandozeilen-Shells auf Linux-Systemen nutzen diese Sprache, um auf Gruppen von Dateien zu verweisen, deren Namen einem bestimmten Muster entsprechen. POSIX.1-2017 spezifiziert die folgenden Musterabgleichszeichen:

`*`
: Entspricht einer beliebigen Anzahl von Zeichen, einschließlich keines Zeichens

`?`
: Entspricht genau einem beliebigen Zeichen

`[]`
: Entspricht einer Klasse von Zeichen

Übersetzt bedeutet das, dass Sie Ihrer Shell vorgeben, ein Muster statt einer genauen Zeichenkette anzunehmen. Normalerweise geben Linux-Benutzer mehrere Dateien mit einem Glob an, anstatt jeden einzelnen Dateinamen einzugeben. Führen Sie die folgenden Befehle aus:

```
$ cd ~/linux_essentials-2.4/globs
$ ls
question1   question14  question2012  star10    star2002
question13  question15  question23    star1100  star2013
$ ls star1*
star10  star1100
$ ls star*
star10  star1100  star2002  star2013
$ ls star2*
star2002  star2013
$ ls star2*2
star2002
$ ls star2013*
star2013
```

Die Shell erweitert `*` zu einer beliebigen Anzahl von Zeichen, so dass `star*` in diesem Kontext alles bedeutet, was mit `star` beginnt. Wenn Sie den Befehl `ls star*` ausführen, führt Ihre Shell das Programm `ls` nicht mit dem Argument `star*` aus — sie sucht vielmehr nach Dateien im aktuellen Verzeichnis, die dem Muster `star*` (einschließlich nur `star`) entsprechen, und verwandelt jede Datei, die dem Muster entspricht, in ein Argument zu `ls`:

```
$ ls star*
```

ist in Bezug auf `ls` dasselbe wie

```
$ ls star10  star1100  star2002  star2013
```

Das Zeichen `*` bedeutet nichts für `ls`. Um dies zu beweisen, führen Sie den folgenden Befehl aus:

```
 ls star\*
ls: cannot access star*: No such file or directory
```

Wenn Sie einem Zeichen ein `\` voranstellen, weisen Sie Ihre Shell an, es nicht zu interpretieren. In diesem Fall möchten Sie, dass `ls` das Argument `star*` hat anstatt den Glob `star*` zu erweitern.

Das `?` erweitert zu einem einzelnen beliebigen Zeichen. Versuchen Sie es selbst mit den folgenden Befehlen:

```
 ls
question1   question14  question2012  star10    star2002
question13  question15  question23    star1100  star2013
$ ls question?
question1
$ ls question1?
question13  question14  question15
$ ls question?3
question13  question23
$ ls question13?
ls: cannot access question13?: No such file or directory
```

Die `[]` Klammern dienen dazu, Bereiche oder Klassen von Zeichen anzugeben. Die `[]` Klammern funktionieren wie in POSIX regulären Ausdrücken, mit der Ausahme, dass bei Globs das `^` anstelle von `!` verwendet wird.

Erstellen Sie einige Dateien, mit denen Sie experimentieren können:

```
 mkdir brackets
$ cd brackets
$ touch file1 file2 file3 file4 filea fileb filec file5 file6 file7
```

Bereiche innerhalb von `[]` Klammern werden mit einem `-` ausgedrückt:

```
 ls
file1  file2  file3  file4  file5  file6  file7  filea  fileb  filec
$ ls file[1-2]
file1  file2
$ ls file[1-3]
file1  file2  file3
```

Es können mehrere Bereiche angegeben werden:

```
 ls file[1-25-7]
file1  file2  file5  file6  file7
$ ls file[1-35-6a-c]
```

```
file1  file2  file3  file5  file6  filea  fileb  filec
```

Eckige Klammern können auch verwendet werden, um einen bestimmten Satz von Zeichen anzugeben:

```
 ls file[1a5]
file1  file5  filea
```

Sie können `^` als erstes Zeichen verwenden, um alles *außer* den folgenden Zeichen anzugeben:

```
 ls file[^a]
file1  file2  file3  file4  file5  file6  file7  fileb  filec
```

Abschließend behandeln wir in dieser Lektion *Zeichenklassen*. Um eine Zeichenklasse anzugeben, verwenden Sie `[:Zeichenklasse:]`. Für die Klasse `digit`, die alle Ziffern umfasst, schreiben Sie beispielweise:

```
$ ls file[[:digit:]]
file1  file2  file3  file4  file5  file6  file7
$ touch file1a file11
$ ls file[[:digit:]a]
file1  file2  file3  file4  file5  file6  file7  filea
$ ls file[[:digit:]]a
file1a
```

Der Glob `file[[:digit:]a]` entspricht `file`, gefolgt von einer Ziffer oder `a`.

Die unterstützten Zeichenklassen hängen von Ihrer aktuellen Locale-Konfiguration ab. POSIX benötigt für alle Locales die folgenden Zeichenklassen:

`[:alnum:]`
: Buchstaben und Zahlen.

`[:alpha:]`
: Groß- oder Kleinbuchstaben.

`[:blank:]`
: Leerzeichen und Tabs.

`[:cntrl:]`
: Steuerzeichen, z.B. Backspace, Glocke, NAK, Escape.

`[:digit:]`
: Zahlen (`0123456789`).

`[:graph:]`
: Alle graphischen Zeichen (alle Zeichen außer `ctrl` und Leerzeichen)

`[:lower:]`
: Kleinbuchstaben (`a-z`).

`[:print:]`
: Druckbare Zeichen (`alnum`, `punct` und das Leerzeichen).

`[:punct:]`
: Interpunktionszeichen, d.h. `!`, `&`, `"`.

`[:space:]`

Whitespace-Zeichen, z.B. Tabs, Leerzeichen, Zeilenumbrüche.

`[:upper:]`
: Großbuchstaben (`A-Z`).

`[:xdigit:]`
: Hexadezimale Zahlen (normalerweise `0123456789abcdefABCDEF`).

Geführte Übungen

1. Gegeben sei die folgende Umgebung. Markieren Sie die Verzeichnisse, die der Befehl `mkdir -p /tmp/outfiles/text/today /tmp/infiles/text/today` erzeugen würde.

```
$ pwd
/tmp
$ find
.
./outfiles
./outfiles/text
```

`/tmp`	
`/tmp/outfiles`	
`/tmp/outfiles/text`	
`/tmp/outfiles/text/today`	
`/tmp/infiles`	
`/tmp/infiles/text`	
`/tmp/infiles/text/today`	

2. Was bewirkt `-v` bei `mkdir`, `rm` und `cp`?
3. Was passiert, wenn Sie versehentlich versuchen, drei Dateien auf der gleichen Befehlszeile in eine bereits vorhandene Datei zu kopieren anstatt in ein Verzeichnis?
4. Was passiert, wenn Sie mit `mv` ein Verzeichnis in sich selbst verschieben?
5. Wie würden Sie alle Dateien in Ihrem aktuellen Verzeichnis löschen, die mit `old` beginnen?
6. Welche der folgenden Dateien würden mit `log_[a-z]_201?_*_01.txt` übereinstimmen?

`log_3_2017_Jan_01.txt`	
`log_+_2017_Feb_01.txt`	
`log_b_2007_Mar_01.txt`	
`log_f_201A_Wednesday_01.txt`	

7. Erstellen Sie ein paar Globs, die der folgenden Liste von Dateinamen entsprechen:

```
doc100
doc200
doc301
doc401
```

Offene Übungen

1. Verwenden Sie die Man Page `cp`, um herauszufinden, wie man eine Kopie einer Datei erstellt und die Berechtigungen und Änderungszeiten mit dem Original übereinstimmen.
2. Was bewirkt der Befehl `rmdir -p`? Experimentieren Sie damit und erklären Sie, wie er sich von `rm -r` unterscheidet.
3. FÜHREN SIE DIESEN BEFEHL NICHT AUS: Was, denken Sie, bewirkt `rm -ri /*`? (EHRLICH, VERSUCHEN SIE NICHT, DAS ZU TUN!)
4. Ist es möglich, außer mit `-i` zu verhindern, dass `mv` Zieldateien überschreibt?
5. Erklären Sie den Befehl `cp -u`.

Zusammenfassung

Die Linux-Befehlszeilenumgebung bietet Werkzeuge zur Verwaltung von Dateien, darunter `cp`, `mv`, `mkdir`, `rm`, `rm` und `rmdir`. Diese Werkzeuge, kombiniert mit Globs, ermöglichen es, sehr schnell viel Arbeit zu erledigen.

Viele Befehle haben die Option `-i`, die Sie vor der Ausführung, den Befehl zu bestätigen. Das sog. *Prompten* kann Ihnen viel Ärger ersparen, wenn Sie sich vertippt haben.

Viele Befehle haben die Option `-r`. In der Mathematik und Informatik ist eine rekursive Funktion eine Funktion, die sich selbst in ihrer Definition verwendet. Wenn es um Kommandozeilen-Tools geht, bedeutet sie in der Regel, den Befehl auf ein Verzeichnis und alles darin anzuwenden.

Befehle, die in dieser Lektion verwendet wurden:

`cat`
: Liest und gibt den Inhalt einer Datei aus.

`cp`
: Kopiert Dateien oder Verzeichnisse.

`echo`
: Gibt eine Zeichenkette aus.

`find`
: Geht durch einen Dateisystembaum und sucht nach Dateien, die einem bestimmten Satz von Kriterien entsprechen.

`ls`
: Zeigt Eigenschaften von Dateien und Verzeichnissen und listet den Inhalt eines Verzeichnisses auf.

`mkdir`
: Erstellt neue Verzeichnisse.

`mv`
: Verschiebt Dateien oder Verzeichnisse und benennt sie um.

`pwd`
: Gibt das aktuelle Arbeitsverzeichnis aus.

`rm`
: Löscht Dateien oder Verzeichnisse.

`rmdir`
: Löscht Verzeichnisse.

`touch`
: Erstellt neue leere Dateien oder aktualisiert den Änderungszeitstempel einer bestehenden Datei.