KAPITEL 4

Ein Ausflug in das Dateisystem

In dem Film *Buckaroo Banzai – Die 8. Dimension*, einem Kultklassiker von 1984, präsentiert uns der verwegene Titelheld folgende Zen-artige Weisheit: »Wohin auch immer du gehst... dort bist du.« Buckaroo hätte durchaus über das Linux-Dateisystem sprechen können:

```
$ cd /usr/share/lib/etc/bin          Wohin auch immer du gehst...
$ pwd
/usr/share/lib/etc/bin               ...dort bist du.
```

Es ist außerdem so, dass Sie, egal wo Sie im Linux-Dateisystem sind – in Ihrem aktuellen Verzeichnis –, irgendwann woanders hingehen werden (in ein anderes Verzeichnis). Je schneller und effizienter Sie diese Navigation erledigen können, umso produktiver werden Sie sein.

Die Techniken in diesem Kapitel helfen Ihnen, schneller und mit weniger Aufwand beim Tippen durch das Dateisystem zu navigieren. Sie sehen täuschend einfach aus, haben dafür aber *enorme* Auswirkungen, eine niedrige Lernkurve und großen Nutzen. Diese Techniken fallen im Großen und Ganzen in zwei Kategorien:

- Sich schnell in ein bestimmtes Verzeichnis bewegen.
- Schnell in ein Verzeichnis zurückkehren, das Sie bereits besucht haben.

Eine schnelle Auffrischung zu Linux-Verzeichnissen finden Sie in Anhang A. Nutzen Sie eine andere Shell als die `bash`, werfen Sie einen Blick in Anhang B.

Bestimmte Verzeichnisse effizient aufsuchen

Wenn Sie zehn Linux-Experten fragen, welcher der lästigste Aspekt der Kommandozeile ist, werden sieben von ihnen sagen: »Das Eintippen der langen Verzeichnispfade.«[1] Falls nämlich Ihre Arbeitsdateien in */home/smith/Work/Projects/Apps/Neutron-Star/src/include* liegen, Ihre Finanzdokumente sich in */home/smith/Finances/Bank/Checking/Statements* befinden und Ihre Videos in */data/Arts/Video/Collection* versammelt sind, ist es überhaupt nicht witzig, immer und immer wieder

1 Ich habe mir das ausgedacht, aber es stimmt trotzdem.

diese Pfade eintippen zu müssen. In diesem Abschnitt lernen Sie Techniken kennen, um effizient zu einem bestimmten Verzeichnis zu gelangen.

Springen Sie in Ihr Home-Verzeichnis

Beginnen wir mit den Grundlagen. Wo auch immer Sie in Ihrem Dateisystem sind, Sie können direkt in Ihr Home-Verzeichnis zurückkehren, indem Sie `cd` ohne Argumente ausführen:

```
$ pwd
/etc                          Sie beginnen irgendwo anders,
$ cd                          führen cd ohne Argumente aus...
$ pwd
/home/smith                   ...und sind wieder zu Hause.
```

Um von irgendeiner Stelle im Dateisystem in Unterverzeichnisse Ihres Home-Verzeichnisses zu springen, bezeichnen Sie Ihr Home-Verzeichnis mit einer Kurzform statt mit einem absoluten Pfad wie */home/smith*. Eine Kurzform ist die Shell-Variable `HOME`:

```
$ cd $HOME/Work
```

Eine andere bietet die Tilde:

```
$ cd ~/Work
```

Sowohl `$HOME` als auch `~` sind Ausdrücke, die von der Shell erweitert werden, eine Tatsache, die Sie überprüfen können, indem Sie sie auf der Standardausgabe anzeigen lassen:

```
$ echo $HOME ~
/home/smith /home/smith
```

Die Tilde kann auch das Home-Verzeichnis eines anderen Benutzers benennen, wenn Sie sie unmittelbar vor dessen Benutzernamen setzen:

```
$ echo ~jones
/home/jones
```

Schneller bewegen mit Tab-Ergänzung

Sparen Sie sich beim Eingeben von `cd`-Befehlen Tipparbeit, indem Sie die Tab-Taste drücken. Dabei werden automatisch Verzeichnisnamen angegeben. Besuchen Sie zum Beispiel mal ein Verzeichnis, das Unterverzeichnisse enthält, wie etwa */usr*:

```
$ cd /usr
$ ls
bin  games  include  lib  local  sbin  share  src
```

Nehmen wir an, Sie wollten das Unterverzeichnis *share* aufsuchen. Tippen Sie `sha` ein und drücken Sie dann einmal die Tab-Taste:

```
$ cd sha<Tab>
```

Die Shell ergänzt den Verzeichnisnamen für Sie:

```
$ cd share/
```

Diese praktische Abkürzung wird *Tab-Ergänzung* genannt. Sie funktioniert sofort, wenn der Text, den Sie eingeben, einem einzigen Verzeichnisnamen entspricht. Passt der Text zu mehreren Verzeichnisnamen, benötigt Ihre Shell weitere Informationen, um den gewünschten Namen zu vervollständigen. Angenommen, Sie hätten nur `s` eingetippt und dann Tab gedrückt:

```
$ cd s<Tab>
```

Die Shell kann den Namen *share* (noch) nicht vervollständigen, weil auch andere Verzeichnisse mit `s` beginnen: *sbin* und *src*. Drücken Sie ein zweites Mal Tab. Die Shell gibt nun alle möglichen Ergänzungen aus, um Ihnen zu helfen:

```
$ cd s<Tab><Tab>
sbin/  share/  src/
```

Dann wartet sie auf Ihre nächste Aktion. Um diese Mehrdeutigkeit aufzulösen, tippen Sie ein weiteres Zeichen ein, `h`, und drücken dann noch einmal Tab:

```
$ cd sh<Tab>
```

Die Shell vervollständigt den Namen des Verzeichnisses für Sie, und aus *sh* wird *share*:

```
$ cd share/
```

Im Allgemeinen drücken Sie Tab einmal, um so viele Ergänzungen wie möglich zu erreichen, oder Sie drücken Tab zweimal, um alle möglichen Ergänzungen auszugeben. Je mehr Zeichen Sie eintippen, umso weniger Mehrdeutigkeit gibt es, und umso besser wird der Treffer.

Die Tab-Ergänzung eignet sich hervorragend zum Beschleunigen der Navigation. Anstatt einen langen Pfad wie */home/smith/Projects/Web/src/include* einzutippen, geben Sie so wenig ein, wie Sie wollen, und drücken einfach immer wieder die Tab-Taste. Mit ein bisschen Übung bekommen Sie den Bogen schnell raus.

Die Tab-Ergänzung variiert je nach Programm

Die Tab-Ergänzung eignet sich nicht nur für `cd`-Befehle. Sie funktioniert für die meisten Befehle, allerdings kann ihr Verhalten variieren. Beim Befehl `cd` ergänzt die Tab-Taste die Verzeichnisnamen. Bei anderen Befehlen, die auf Dateien operieren, wie `cat`, `grep` und `sort`, erweitert die Tab-Ergänzung auch Dateinamen. Lautet der Befehl `ssh` (Secure Shell), vervollständigt sie Hostnamen. Beim Befehl `chown` (Ändern des Besitzers einer Datei) werden Benutzernamen vervollständigt. Sie können sogar Ihre eigenen Ergänzungsregeln herstellen, um schneller zu werden, wie wir in Beispiel 4-1 sehen werden. Schauen Sie außerdem auch in `man bash` und lesen Sie den Abschnitt »programmable completion«.

Mit Aliasen oder Variablen in oft besuchte Verzeichnisse springen

Sollten Sie häufig ein weit entfernt liegendes Verzeichnis besuchen, wie etwa */home/smith/Work/?Projects?/Web/src/include*, legen Sie sich am besten einen Alias an, der die `cd`-Operation ausführt:

```
# In einer Shell-Konfigurationsdatei:
alias work="cd $HOME/Work/Projects/Web/src/include"
```

Führen Sie einfach jedes Mal den Alias aus, wenn Sie Ihr Ziel erreichen wollen:

```
$ work
$ pwd
/home/smith/Work/Projects/Web/src/include
```

Stellen Sie alternativ eine Variable für den Verzeichnispfad her:

```
$ work=$HOME/Work/Projects/Web/src/include
$ cd $work
$ pwd
/home/smith/Work/Projects/Web/src/include
$ ls $work/
css                                 Benutzt die Variable auf andere Arten.
main.css  mobile.css
```

Häufig bearbeitete Dateien mit einem Alias bearbeiten

Manchmal liegt der Grund für das häufige Aufsuchen eines Verzeichnisses darin, dass Sie eine bestimmte Datei bearbeiten müssen. Falls das der Fall ist, sollten Sie einen Alias für das Bearbeiten dieser Datei mit dem absoluten Pfad definieren, ohne das Verzeichnis zu wechseln. Die folgende Alias-Definition erlaubt Ihnen, *$HOME/.bashrc* zu bearbeiten – ganz egal, wo Sie sich im Dateisystem befinden –, indem Sie `rcedit` ausführen. Es ist kein `cd` erforderlich:

```
# Setzen Sie dies in eine Shell-Konfigurationsdatei
# und laden Sie sie neu:
alias rcedit='$EDITOR $HOME/.bashrc'
```

Falls Sie regelmäßig viele Verzeichnisse mit langen Pfadnamen besuchen, können Sie Aliase oder Variablen für sie alle herstellen. Das bringt jedoch einige Nachteile mit sich:

- Es ist schwer, sich diese ganzen Aliase/Variablen zu merken.
- Sie könnten versehentlich einen Alias erzeugen, der denselben Namen trägt wie ein vorhandener Befehl, was einen Konflikt erzeugt.

Eine Alternative besteht darin, eine Shell-Funktion wie in Beispiel 4-1 herzustellen, die ich `qcd` (*quick cd*) genannt habe. Diese Funktion nimmt einen String-Schlüssel als Argument entgegen, wie `work` oder `recipes`, und führt `cd` auf einem ausgewählten Verzeichnispfad aus.

Beispiel 4-1: Eine Funktion für den Wechsel in weit entfernte Verzeichnisse mit cd

```
# Definieren der Funktion qcd
qcd () {
  # Nimmt 1 Argument entgegen, das ein String-Schlüssel ist, und
  # führt für jeden Schlüssel eine andere "cd"-Operation aus.
  case "$1" in
    work)
      cd $HOME/Work/Projects/Web/src/include
      ;;
    recipes)
      cd $HOME/Family/Cooking/Recipes
      ;;
    video)
      cd /data/Arts/Video/Collection
      ;;
    beatles)
      cd $HOME/Music/mp3/Artists/B/Beatles
      ;;
    *)
      # Das angegebene Argument gehörte nicht zu den unterstützten Schlüsseln.
      echo "qcd: unknown key '$1'"
      return 1
      ;;
  esac
  # Der aktuelle Verzeichnisname wird ausgegeben, damit man weiß, wo man ist.
  pwd
}
# Einrichten einer Tab-Ergänzung
complete -W "work recipes video beatles" qcd
```

Speichern Sie die Funktion in einer Shell-Konfigurationsdatei wie *$HOME/.bashrc* (siehe »Umgebungen und Initialisierungsdateien, die Kurzfassung« auf Seite 49) und laden Sie diese neu. Damit ist sie bereit für den Einsatz. Tippen Sie `qcd` gefolgt von einem der unterstützten Schlüssel ein, um schnell in das damit verknüpfte Verzeichnis zu springen:

```
$ qcd beatles
/home/smith/Music/mp3/Artists/B/Beatles
```

Als Bonus führt die letzte Zeile des Skripts den Befehl `complete` aus, ein Shell-Builtin, das eine angepasste Tab-Ergänzung für `qcd` einrichtet, damit es die vier unterstützten Schlüssel vervollständigt. Nun müssen Sie sich die Argumente von `qcd` nicht mehr merken! Tippen Sie einfach `qcd` gefolgt von einem Leerzeichen ein und drücken Sie dann zweimal die Tab-Taste. Die Shell gibt nun alle Schlüssel für Ihre Referenz aus, und Sie können den gewünschten Schlüssel auf die übliche Weise vervollständigen:

```
$ qcd <Tab><Tab>
beatles  recipes  video    work
$ qcd v<Tab><Enter>                    Ergänzt 'v' zu 'video'.
/data/Arts/Video/Collection
```

Machen Sie ein großes Dateisystem gefühlt kleiner mit CDPATH

Die qcd-Funktion verarbeitet nur die Verzeichnisse, die Sie angeben. Die Shell bietet eine allgemeinere cd-Lösung ohne diese Einschränkung, den sogenannten *cd-Suchpfad*. Seit ich diese Shell-Eigenschaft kenne, navigiere ich ganz anders im Linux-Dateisystem.

Nehmen wir einmal an, Sie hätten ein wichtiges Unterverzeichnis namens *Photos*, das Sie oft besuchen. Es liegt unter */home/smith/Family/Memories/Photos*. Wenn Sie sich im Dateisystem umherbewegen, müssen Sie jedes Mal einen langen Pfadnamen eintippen, um *Photos* zu besuchen:

```
$ cd ~/Family/Memories/Photos
```

Wäre es nicht großartig, wenn Sie diesen Pfad auf *Photos* abkürzen könnten, ganz egal, wo Sie sich im Dateisystem gerade befinden?

```
$ cd Photos
```

Normalerweise funktioniert dieser Befehl nicht:

```
bash: cd: Photos: No such file or directory
```

Es sei denn, Sie befinden sich gerade im richtigen Parent-Verzeichnis (*~/Family/Memories*) oder in einem anderen Verzeichnis, das zufällig ebenfalls ein Unterverzeichnis mit dem Namen *Photos* enthält. Nun, mit ein paar kleinen Vorbereitungen können Sie cd anweisen, auch an anderen Orten als Ihrem aktuellen Verzeichnis nach dem *Photos*-Unterverzeichnis zu suchen. Die Suche erfolgt blitzschnell und in allen Parent-Verzeichnissen, die Sie angeben. Sie könnten zum Beispiel cd anweisen, zusätzlich zu Ihrem aktuellen Verzeichnis auch das Verzeichnis *$HOME/Family/Memories* zu durchsuchen. Wenn Sie dann an irgendeiner anderen Stelle im Dateisystem `cd Photos` eintippen, wird cd erfolgreich sein:

```
$ pwd
/etc
$ cd Photos
/home/smith/Family/Memories/Photos
```

Ein cd-Suchpfad funktioniert genauso wie der Befehlssuchpfad, $PATH, allerdings werden anstelle von Befehlen Unterverzeichnisse gefunden. Konfigurieren Sie ihn mit der Shell-Variablen CDPATH, die dasselbe Format hat wie PATH: eine Liste von Verzeichnissen, die durch Doppelpunkte getrennt sind. Falls Ihr CDPATH zum Beispiel aus diesen vier Verzeichnissen besteht:

```
$HOME:$HOME/Projects:$HOME/Family/Memories:/usr/local
```

und Sie ...

```
$ cd Photos
```

eintippen, prüft cd nacheinander, ob die folgenden Verzeichnisse existieren, bis es ein passendes findet oder komplett scheitert:

1. *Photos* im aktuellen Verzeichnis
2. *$HOME/Photos*
3. *$HOME/Projects/Photos*
4. *$HOME/Family/Memories/Photos*
5. */usr/local/Photos*

In diesem Fall ist `cd` beim vierten Versuch erfolgreich und ändert das Verzeichnis auf *$HOME/Family/Memories/Photos*. Wenn zwei Verzeichnisse in `$CDPATH` ein Unterverzeichnis namens *Photos* besitzen, gewinnt das Parent-Verzeichnis, das zuerst kommt.

Hinweis

Normalerweise liefert ein erfolgreiches `cd` keine Ausgabe. Wenn `cd` jedoch mithilfe Ihres `CDPATH` ein Verzeichnis findet, gibt es den absoluten Pfad auf der Standardausgabe aus, um Sie über das neue aktuelle Verzeichnis in Kenntnis zu setzen:

```
$ CDPATH=/usr   Setzt einen CDPATH.
$ cd /tmp       Keine Ausgabe: CDPATH wurde nicht befragt.
$ cd bin        cd befragt CDPATH...
/usr/
bin          ...und gibt das neue Arbeitsverzeichnis aus.
```

Setzen Sie Ihre wichtigsten oder am häufigsten benutzten Parent-Verzeichnisse in `CDPATH`, und Sie können mit `cd` von beliebigen Stellen in Ihrem Dateisystem aus in deren Unterverzeichnisse springen, ohne den kompletten Pfad eintippen zu müssen. Vertrauen Sie mir, das ist *erstaunlich*. Die folgende Fallstudie sollte das beweisen.

Organisieren Sie Ihr Home-Verzeichnis für eine schnelle Navigation

Benutzen wir `CDPATH`, um die Art und Weise zu vereinfachen, wie Sie zu Ihrem Home-Verzeichnis navigieren. Mit einer kleinen Konfiguration können Sie viele Verzeichnisse innerhalb Ihres Home-Verzeichnisses leicht zugänglich machen, ohne viel tippen zu müssen, und das unabhängig davon, wo Sie sich in Ihrem Dateisystem befinden. Diese Technik funktioniert am besten, wenn Ihr Home-Verzeichnis gut organisiert ist und wenigstens zwei Ebenen an Unterverzeichnissen besitzt. Abbildung 4-1 zeigt ein Beispiel für ein gut organisiertes Verzeichnislayout.

Der Trick besteht darin, Ihren `CDPATH` so einzurichten, dass er in der angegebenen Reihenfolge Folgendes enthält:

1. *$HOME*
2. Ihre Auswahl an Unterverzeichnissen von *$HOME*
3. den relativen Pfad für ein Parent-Verzeichnis, gekennzeichnet mithilfe zweier Punkte (`..`)

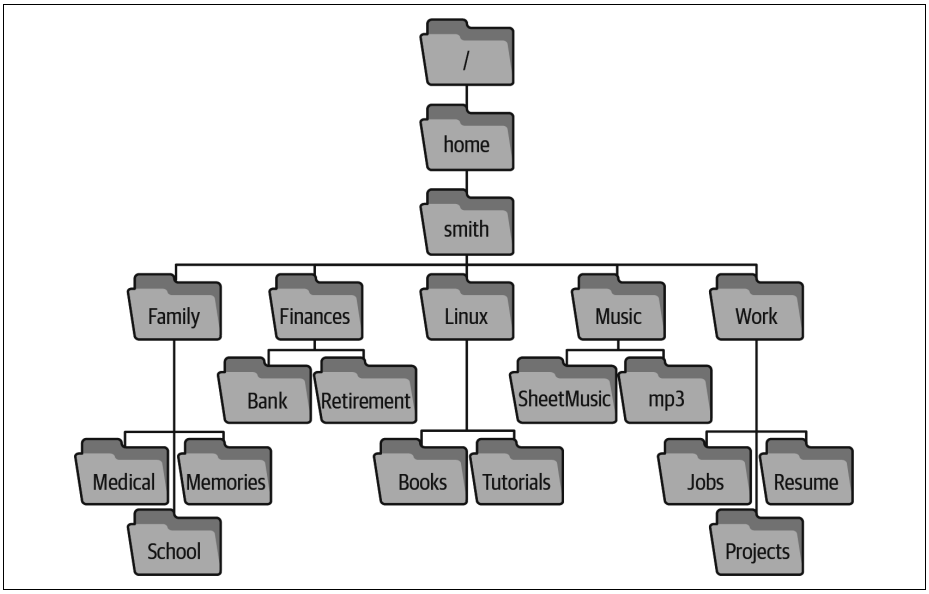

Abbildung 4-1: Zwei Ebenen an Unterverzeichnissen im Verzeichnis /home/smith

Indem Sie *$HOME* einbeziehen, können Sie unmittelbar in eines von seinen Unterverzeichnissen springen (*Family*, *Finances*, *Linux*, *Music* und *Work*), ohne dass Sie den kompletten Pfad angeben müssen, auch wenn Sie sich an irgendeiner anderen Stelle im Dateisystem befinden:

```
$ pwd
/etc                          Beginnt außerhalb Ihres Home-Verzeichnisses.
$ cd Work
/home/smith/Work
$ cd Family/School            Sie sind 1 Ebene unter $HOME gesprungen.
/home/smith/Family/School
```

Durch das Einfügen der Unterverzeichnisse von *$HOME* in Ihren `CDPATH` können Sie auf einen Schlag in *deren* Unterverzeichnisse springen:

```
$ pwd
/etc                          Irgendwo außerhalb Ihres Home-Verzeichnisses.
$ cd School
/home/smith/Family/School     Sie sind 2 Ebenen unter $HOME gesprungen.
```

Alle Verzeichnisse in Ihrem `CDPATH` sind bisher absolute Pfade in *$HOME* und dessen Unterverzeichnissen. Da Sie jedoch auch den relativen Pfad `..` aufgenommen haben, ermöglichen Sie ein neues `cd`-Verhalten in *jedem* Verzeichnis. Ganz egal wo Sie im Dateisystem sind, Sie können anhand des Namens in jedes *Sibling*-Verzeichnis (*../sibling*) springen, ohne die zwei Punkte eintippen zu müssen, weil `cd` Ihren aktuellen Parent durchsucht. Falls Sie sich zum Beispiel in */usr/bin* befinden und sich nach */usr/lib* begeben wollen, benötigen Sie nur `cd lib`:

```
$ pwd
/usr/bin                              Ihr aktuelles Verzeichnis
$ ls ..
bin   include   lib   src             Ihre Siblings
$ cd lib
/usr/lib                              Sie sind zu einem Sibling gesprungen.
```

Oder sollten Sie ein Programmierer sein, der an Code arbeitet, der die Unterverzeichnisses *rc*, *include* und *docs* hat:

```
$ pwd
/usr/src/myproject
$ ls
docs   include   src
```

können Sie zügig zwischen den Unterverzeichnissen hin- und herspringen:

```
$ cd docs                             Wechseln Sie Ihr aktuelles Verzeichnis.
$ cd include
/usr/src/myproject/include            Sie sind zu einem Sibling gesprungen.
$ cd src
/usr/src/myproject/src                Noch einmal.
```

Ein CDPATH für den Baum in Abbildung 4-1 könnte sechs Einträge enthalten: Ihr Home-Verzeichnis, vier seiner Unterverzeichnisse und den relativen Pfad für ein Parent-Verzeichnis:

```
# Setzen Sie das in eine Shell-Konfigurationsdatei und laden Sie diese:
export CDPATH=$HOME:$HOME/Work:$HOME/Family:$HOME/Linux:$HOME/Music:..
```

Nachdem Sie die Konfigurationsdatei geladen haben, können Sie mit cd in viele wichtige Verzeichnisse springen, ohne lange Verzeichnispfade eintippen zu müssen. Es sind nun nur noch die kurzen Verzeichnisnamen nötig. Hurra!

Diese Technik funktioniert am besten, wenn alle Unterverzeichnisse unter den CDPATH-Verzeichnissen eindeutige Namen haben. Sollten sich Namen wiederholen, wie etwa bei *$HOME/Music* und *$HOME/Linux/Music*, erzielen Sie möglicherweise nicht das gewünschte Verhalten. Der Befehl cd Music prüft immer *$HOME* vor *$HOME/Linux* und findet deshalb *$HOME/Linux/Music* bei der Suche nicht.

Um in den ersten zwei Ebenen von *$HOME* nach doppelt auftretenden Unterverzeichnisnamen zu suchen, probieren Sie diesen frechen Einzeiler aus. Er listet alle Unter- und Unterunterverzeichnisse von *$HOME* auf, isoliert die Unterunterverzeichnisnamen mit cut, sortiert die Liste und zählt die Vorkommen mit uniq:

```
$ cd
$ ls -d */ && (ls -d */*/ | cut -d/ -f2-) | sort | uniq -c | sort -nr | less
```

Sie kennen diese Technik zum Prüfen auf Duplikate vielleicht aus »Dateiduplikate entdecken« auf Seite 33. Wenn die Ausgabe eine Zahl anzeigt, die größer als 1 ist, haben Sie Duplikate. Ich weiß, dass dieser Befehl einige Dinge enthält, die wir noch nicht behandelt haben. Sie lernen den doppelten Ampersand (&&) in »Technik #1: Bedingte Listen« auf Seite 130 und die runden Klammern in »Technik #10: Explizite Subshells« auf Seite 151 kennen.

Effizient zu Verzeichnissen zurückkehren

Gerade haben Sie gelernt, wie Sie effizient ein Verzeichnis besuchen. Jetzt zeige ich Ihnen, wie Sie schnell wieder zu einem Verzeichnis zurückkehren können.

Mit »cd -« zwischen zwei Verzeichnissen umschalten

Nehmen wir an, Sie arbeiteten in einem ganz tief gelegenen Verzeichnis und führten `cd` aus, um woandershin zu gehen:

```
$ pwd
/home/smith/Finances/Bank/Checking/Statements
$ cd /etc
```

Und plötzlich denken Sie: »Nein, warte, ich will zurück in das Verzeichnis *Statements*, in dem ich gerade war.« Tippen Sie auf keinen Fall den langen Verzeichnispfad ein, sondern führen Sie einfach `cd` mit einem Bindestrich als Argument aus:

```
$ cd -
/home/smith/Finances/Bank/Checking/Statements
```

Dieser Befehl bringt Ihre Shell in das vorhergehende Verzeichnis zurück und gibt dankenswerterweise auch noch dessen absoluten Pfad aus, damit Sie wissen, wo Sie sind.

Um zwischen zwei Verzeichnissen hin- und herzuspringen, führen Sie wiederholt `cd -` aus. Das spart unglaublich viel Zeit, wenn Sie in einer einzigen Shell in zwei Verzeichnissen gleichzeitig arbeiten müssen. Es gibt allerdings einen Haken: Die Shell erinnert sich immer nur an ein vorhergehendes Verzeichnis. Wenn Sie zum Beispiel zwischen */usr/local/bin* und */etc* hin- und herwechseln:

```
$ pwd
/usr/local/bin
$ cd /etc                  Die Shell erinnert sich an /usr/local/bin
$ cd -                     Die Shell erinnert sich an /etc
/usr/local/bin
$ cd -                     Die Shell erinnert sich an /usr/local/bin
/etc
```

und `cd` ohne Argumente aufrufen, um in Ihr Home-Verzeichnis zu springen:

```
$ cd                       Die Shell erinnert sich an /etc
```

hat die Shell */usr/local/bin* als vorheriges Verzeichnis vergessen:

```
$ cd -                     Die Shell erinnert sich an Ihr Home-Verzeichnis
/etc
$ cd -                     Die Shell erinnert sich an /etc
/home/smith
```

Die nächste Technik überwindet diese Einschränkung.

Mit pushd und popd zwischen vielen Verzeichnissen wechseln

Der Befehl `cd -` schaltet zwischen zwei Verzeichnissen hin und her. Doch was ist, wenn Sie drei oder mehr Verzeichnisse im Blick behalten wollen? Nehmen wir einmal an, Sie legten auf Ihrem Linux-Computer eine lokale Website an. Diese Aufgabe erfordert oft vier oder mehr Verzeichnisse:

- den Ort der live geschalteten, freigegebenen Webseiten, wie etwa */var/www/html*,
- das Konfigurationsverzeichnis des Webservers, oft */etc/apache2*,
- den Ort der SSL-Zertifikate, oft */etc/ssl/certs* sowie
- Ihr Arbeitsverzeichnis, also etwa *~/Work/Projects/Web/src*.

Glauben Sie mir, es ist mühselig, immer wieder zu tippen:

```
$ cd ~/Work/Projects/Web/src
$ cd /var/www/html
$ cd /etc/apache2
$ cd ~/Work/Projects/Web/src
$ cd /etc/ssl/certs
```

Wenn Sie einen großen Bildschirm haben, können Sie sich die Strapazen ersparen, indem Sie für jedes Verzeichnis ein eigenes Shell-Fenster öffnen. Falls Sie aber in einer einzigen Shell arbeiten (zum Beispiel über eine SSH-Verbindung), sollten Sie eine Shell-Eigenschaft namens *Verzeichnis-Stack* ausnutzen. Dieser erlaubt Ihnen, ganz leicht durch mehrere Verzeichnisse zu reisen. Dazu verwendet er die eingebauten Shell-Befehle `pushd`, `popd` und `dirs`. Es dauert vielleicht eine Viertelstunde, das zu erlernen, und der Vorteil, den Sie damit erwerben, bleibt Ihnen für den Rest Ihrer Karriere erhalten.[2]

Ein *Verzeichnis-Stack* ist eine Liste von Verzeichnissen, die Sie in der aktuellen Shell besucht haben und die Sie weiter im Auge behalten wollen. Sie manipulieren den Stack, indem Sie zwei Operationen ausführen, die *Push* und *Pop* genannt werden. Der Push eines Verzeichnisses fügt dieses am Anfang der Liste hinzu, die traditionell als Anfang oder *Top* des Stacks bezeichnet werden.[3] Ein Pop entfernt das oberste Verzeichnis aus dem Stack. Zu Anfang enthält der Stack nur Ihr aktuelles Verzeichnis, aber Sie können Verzeichnisse hinzufügen (Push) und entfernen (Pop) sowie mit `cd` zwischen ihnen wechseln.

Hinweis
Jede laufende Shell führt ihren eigenen Verzeichnis-Stack.

2 Eine Alternative wäre, mithilfe von Kommandozeilenprogrammen wie `screen` und `tmux` mehrere virtuelle Displays zu öffnen. Diese Programme werden als *Terminal-Multiplexer* bezeichnet. Sie erfordern einen größeren Lernaufwand als Verzeichnis-Stacks, lohnen die Betrachtung aber ebenfalls.

3 Stacks werden in der Informatik auch als Stapel benannt, und genau so sollten Sie sich das auch vorstellen: Sie legen etwas oben auf einem Stapel ab und können auch nur von oben etwas herunternehmen. Was zuletzt auf den Stapel gepackt wurde, muss zuerst wieder weggenommen werden.

Ich beginne mit den Grundoperationen (Push, Pop, Betrachten) und komme dann zu dem guten Kram.

Ein Verzeichnis auf den Stack schieben (Push)

Der Befehl `pushd` (kurz für *push directory*) macht Folgendes:

1. Er fügt ein bestimmtes Verzeichnis oben dem Stack hinzu.
2. Er führt ein `cd` zu diesem Verzeichnis aus.
3. Er gibt den Stack von oben bis unten aus, damit Sie einen Vergleich haben.

Ich werde einen Verzeichnis-Stack aus vier Verzeichnissen aufbauen, indem ich diese nacheinander auf den Stack schiebe:

```
$ pwd
/home/smith/Work/Projects/Web/src
$ pushd /var/www/html
/var/www/html ~/Work/Projects/Web/src
$ pushd /etc/apache2
/etc/apache2 /var/www/html ~/Work/Projects/Web/src
$ pushd /etc/ssl/certs
/etc/ssl/certs /etc/apache2 /var/www/html ~/Work/Projects/Web/src
$ pwd
/etc/ssl/certs
```

Nach jeder `pushd`-Operation gibt die Shell den Stack aus. Das aktuelle Verzeichnis ist das ganz linke (oberste) Verzeichnis.

Betrachten eines Verzeichnis-Stacks

Sie geben den Verzeichnis-Stack einer Shell mit dem Befehl `dirs` aus. Dabei wird der Stack nicht verändert:

```
$ dirs
/etc/ssl/certs /etc/apache2 /var/www/html ~/Work/Projects/Web/src
```

Falls Sie den Stack von oben nach unten anschauen wollen, nutzen Sie die Option `-p`:

```
$ dirs -p
/etc/ssl/certs
/etc/apache2
/var/www/html
~/Work/Projects/Web/src
```

Sie können die Ausgabe sogar mit einer Pipeline auf den Befehl `nl` leiten, um die Zeilen mit null beginnend zu nummerieren:

```
$ dirs -p | nl -v0
     0  /etc/ssl/certs
     1  /etc/apache2
     2  /var/www/html
     3  ~/Work/Projects/Web/src
```

Noch einfacher: Führen Sie `dirs -v` aus, um den Stack mit nummerierten Zeilen auszugeben:

```
$ dirs -v
 0  /etc/ssl/certs
 1  /etc/apache2
 2  /var/www/html
 3  ~/Work/Projects/Web/src
```

Falls Sie dieses Top-down-Format künftig immer benutzen wollen, könnten Sie sich dafür einen Alias anlegen:

```
# Setzen Sie dies in eine Shell-Konfigurationsdatei und laden Sie sie:
alias dirs='dirs -v'
```

Ein Verzeichnis vom Stack holen (Pop)

Der Befehl `popd` (kurz für *pop directory*) ist das Gegenstück zu `pushd`. Er macht Folgendes:

1. Er entfernt das oberste Verzeichnis vom Stack.
2. Er führt ein `cd` zum neuen obersten Verzeichnis durch.
3. Er gibt zu Ihrer Information den Stack von oben nach unten aus.

Falls Ihr Stack zum Beispiel vier Verzeichnisse enthält:

```
$ dirs
/etc/ssl/certs /etc/apache2 /var/www/html ~/Work/Projects/Web/src
```

wird ein wiederholtes Ausführen von `popd` die Verzeichnisse von oben bis unten durchlaufen:

```
$ popd
/etc/apache2 /var/www/html ~/Work/Projects/Web/src
$ popd
/var/www/html ~/Work/Projects/Web/src
$ popd
~/Work/Projects/Web/src
$ popd
bash: popd: directory stack empty
$ pwd
~/Work/Projects/Web/src
```

Tipp

Die Befehle `pushd` und `popd` sind so effektiv, dass ich Ihnen empfehle, Aliase aus zwei Buchstaben für sie anzulegen, damit sie genauso schnell getippt werden können wie `cd`:

```
# Setzen Sie das in eine Shell-Konfigurationsdatei
# und laden Sie sie:
alias gd=pushd
alias pd=popd
```

Verzeichnisse im Stack vertauschen

Sie können nun also den Verzeichnis-Stack aufbauen und wieder leeren. Schauen wir uns jetzt einige praktische Anwendungsfälle an. `pushd` ohne Argumente vertauscht die obersten beiden Verzeichnisse und navigiert zum neuen obersten Verzeichnis. Springen wir mehrmals zwischen */etc/apache2* und Ihrem Arbeitsverzeichnis hin und her, indem wir einfach `pushd` ausführen. Sie sehen, dass das dritte Verzeichnis */var/www/html* an seiner Position im Stack bleibt, während die ersten beiden Verzeichnisse die Positionen tauschen:

```
$ dirs
/etc/apache2 ~/Work/Projects/Web/src /var/www/html
$ pushd
~/Work/Projects/Web/src /etc/apache2 /var/www/html
$ pushd
/etc/apache2 ~/Work/Projects/Web/src /var/www/html
$ pushd
~/Work/Projects/Web/src /etc/apache2 /var/www/html
```

Der Befehl `pushd` verhält sich ähnlich wie `cd -`, wechselt also zwischen zwei Verzeichnissen, allerdings hat er nicht die Beschränkung, sich immer nur ein Verzeichnis merken zu können.

Verwandeln Sie ein versehentliches cd in ein pushd

Nehmen wir einmal an, dass Sie mit `pushd` zwischen mehreren Verzeichnissen hin und her springen. Plötzlich führen Sie versehentlich stattdessen `cd` aus und verlieren ein Verzeichnis:

```
$ dirs
~/Work/Projects/Web/src /var/www/html /etc/apache2
$ cd /etc/ssl/certs
$ dirs
/etc/ssl/certs /var/www/html /etc/apache2
```

Hoppla, der versehentliche `cd`-Befehl hat *~/Work/Projects/Web/src* im Stack durch */etc/ssl/certs* ersetzt. Aber keine Panik. Sie können das verloren gegangene Verzeichnis wieder auf den Stack packen, ohne dessen langen Pfad eintippen zu müssen. Führen Sie zweimal `pushd` aus, einmal mit dem Bindestrich als Argument und einmal ohne:

```
$ pushd -
~/Work/Projects/Web/src /etc/ssl/certs /var/www/html /etc/apache2
$ pushd
/etc/ssl/certs ~/Work/Projects/Web/src /var/www/html /etc/apache2
```

Schauen wir uns genau an, wieso das funktioniert:

- Das erste `pushd` kehrt zum vorhergehenden Verzeichnis Ihrer Shell, *~/Work/Projects/Web/src*, zurück und schiebt es auf den Stack. `pushd` akzeptiert genau wie `cd` einen Bindestrich als Argument, der bedeutet: »Gehe zurück in mein vorheriges Verzeichnis.«

- Der zweite `pushd`-Befehl vertauscht die beiden oberen Verzeichnisse, sodass Sie zurück zu */etc/ssl/certs* gelangen. Sie haben also nun *~/Work/Projects/Web/src* wieder an die zweite Position im Stack befördert, also genau dorthin, wo es gewesen wäre, wenn Sie den Fehler nicht gemacht hätten.

Dieser »Hoppla, ich habe ein `pushd` vergessen«-Befehl ist so nützlich, dass er einen eigenen Alias verdient. Ich selbst nenne ihn `slurp` (schlürfen), weil er (zumindest in meiner Vorstellung) ein Verzeichnis »zurückschlürft«, das ich versehentlich verloren habe:

```
# Setzen Sie dies in eine Shell-Konfigurationsdatei und laden Sie sie:
alias slurp='pushd - && pushd'
```

Tiefer in den Stack eintauchen

Was ist, wenn Sie mit `cd` zwischen anderen als den obersten beiden Verzeichnissen im Stack wechseln wollen? `pushd` und `popd` akzeptieren ein positives oder negatives Integer-Argument, um weiter in den Stack einzutauchen. Der Befehl:

```
$ pushd +N
```

verschiebt *N* Verzeichnisse vom Anfang des Stacks (also von oben) an das Ende (also nach unten) und führt dann ein `cd` an das neue oberste Verzeichnis aus. Ein negatives Argument (*-N*) verschiebt Verzeichnisse in die entgegengesetzte Richtung, also von unten nach oben, bevor `cd` ausgeführt wird.[4]

```
$ dirs
/etc/ssl/certs ~/Work/Projects/Web/src /var/www/html /etc/apache2
$ pushd +1
~/Work/Projects/Web/src /var/www/html /etc/apache2 /etc/ssl/certs
$ pushd +2
/etc/apache2 /etc/ssl/certs ~/Work/Projects/Web/src /var/www/html
```

Auf diese Weise können Sie mit einem einfachen Befehl zu jedem anderen Verzeichnis im Stack springen. Falls Ihr Stack jedoch sehr lang ist, könnte es schwierig sein, die numerische Position eines bestimmten Verzeichnisses abzuschätzen. Geben Sie also mit `dirs -v` diese numerischen Positionen aus, wie Sie es in »Betrachten eines Verzeichnis-Stacks« auf Seite 80 gemacht haben:

```
$ dirs -v
 0  /etc/apache2
 1  /etc/ssl/certs
 2  ~/Work/Projects/Web/src
 3  /var/www/html
```

Um */var/www/html* im Stack ganz nach oben zu verschieben (und damit zu Ihrem aktuellen Verzeichnis zu machen), führen Sie `pushd +3` aus.

Um zum Verzeichnis ganz unten im Stack zu springen, führen Sie `pushd -0` (Bindestrich null) aus:

4 Programmierer kennen diese Operationen vermutlich als das Rotieren des Stacks.

```
$ dirs
/etc/apache2 /etc/ssl/certs ~/Work/Projects/Web/src /var/www/html
$ pushd -0
/var/www/html /etc/apache2 /etc/ssl/certs ~/Work/Projects/Web/src
```

Sie können auch Verzeichnisse jenseits des obersten Verzeichnisses aus dem Stack entfernen, indem Sie `popd` mit einem numerischen Argument benutzen. Der Befehl:

```
$ popd +N
```

entfernt das Verzeichnis an Position *N* aus dem Stack. Gezählt wird hierbei von oben. Ein negatives Argument (`-N`) zählt stattdessen von unten im Stack nach oben. Das Zählen beginnt bei null; `popd +1` entfernt also das zweite Verzeichnis von oben:

```
$ dirs
/var/www/html /etc/apache2 /etc/ssl/certs ~/Work/Projects/Web/src
$ popd +1
/var/www/html /etc/ssl/certs ~/Work/Projects/Web/src
$ popd +2
/var/www/html /etc/ssl/certs
```

Zusammenfassung

Alle Techniken in diesem Kapitel sind mit ein bisschen Übung leicht zu verstehen und ersparen Ihnen eine Menge Zeit und Arbeit beim Tippen. Ich persönlich finde diese Techniken besonders nützlich:

- `CDPATH` für eine schnelle Navigation.
- `pushd` und `popd` für ein schnelles Zurückkehren.
- Gelegentlich auch einmal `cd -`.