Skript
Unix-Crashkurs HU Berlin

19.10.2021

Inhaltsverzeichnis

1 Über dieses Skript

Dieses Skript ist zwar eine Verschriftlichung des Kurses, entspricht aber nicht 1:1 den Inhalten. Somit eignet es sich eher als zusätzliche/weitere Erklärung als als Ersatz für Teilnahme an den Terminen der Veranstaltung.

2 Einführung

Unix ist ein multitasking- und mehrbenutzerfähiges Betriebssystem für Computersysteme. Die Entwicklung begann in den 1970er Jahren an den Bell Labs, unter anderem durch Dennis Ritchie, der unter anderem auch die Programmiersprache C entwickelt hat. [1] Ziel von Unix war ein Betriebssystem als Grundlage für die Entwicklung auf Systemen in den Bell Labs, jedoch wurde gegen Ende der 1970er Jahre durch die damalige Mutterfirma AT&T das Betriebssystem auch an weitere Firmen und Universitäten lizenziert, wodurch eine Vielzahl an Versionen entstand.
In der ersten Hälfte der 1990er Jahre verkaufte AT&T die Rechte an UNIX an die Firma Novell, welche dann in 1995 ihr komplettes UNIX-Geschäft an die SCO verkaufte. [2]
Unix-artige Betriebssysteme zeichnen sich durch ein modulares Design aus, sie verfügen über ein hierarchisches, baumartiges Dateisystem, sowie über Möglichkeit der Interprozess-Kommunikation.
Das ursprüngliche Unix und viele der kommerziellen Derivate, die auf einer der Lizenz von

AT&T oder deren Nachfolger als Lizenzinhaber basieren, sind Closed-Source und oftmals ist nicht nur die Nutzung durch, sondern auch die Weiterverteilung an Nichtinhaber der Lizenz durch die Lizenzbedingungen verboten. Bei dem heutzutage weit verbreiteten Linux handelt es sich um einen Unix-artigen Kernel der als Open-Source-Software das erste mal am 17. September 1991 von Linus Torvalds als Projekt während seiner Studienzeit an der Universität Helsinki veröffentlicht. [3] Torvalds entwickelte den Linux-Kernel, da ihm die Lizenzbedingungen des Minix-Linux nicht gefielen. Linux betreibt rund 77 % aller Webserver, betreibt nahezu alle TOP500-Supercomputer, als Grundlage von Android rund 87 % aller Smartphones und ist auch im Universitätsbetrieb weit verbreitet. [4] [5] [6] Nur im Endnutzerbereich auf Arbeitsplatzrechnern konnte es nie so recht Fuß fassen und hat dort rund 5 % Verbreitung, Unix-artige Betriebssystem kommen hier insgesamt auf rund 20 %, die zum größten Teil auf MacOS zurückzuführen sind, welches auf BSD basiert. [7]
Ob es sich bei Linux nur um den Betriebssystemkern oder das gesamte Betriebssystem handelt wird oft diskutiert, da Linux fast ausschließlich in einer Distribution verteilt und genutzt wird. Die heutige weite Verbreitung von Linux rührt auch daher, dass Linux im Gegensatz zum ursprünglichen Unix Open-Source ist, was bedeutet, dass der Nutzer Einblick in den Quelltext des Betriebssystems nehmen kann. Somit steht einer Weiterverbreitung des Betriebsysstems oder sogar einer Weiterentwicklung nichts entgegen. Im kommerziellen Umfeld werden daher oft Distributionen mit einem Servicevertrag eingesetzt, bei dem der Distributionsverwalter sich verpflichtet Fehler oder Sicherheitslücken in bestimmten Zeiträumen zu beheben oder Umgehungsstrategien zu liefern. Zu den großen Distributionen mit Firmensupport gehören Red Hat Enterprise Linux, SUSE Linux Enterprise Server und Canoncial Ubuntu.

3 Motivation

Wie im vorherigen Abschnitt angeführt stellt Unix die Grundlage vieler EDV-Geräte dar und wird auch an der Humboldt Universität in der Informatik eingesetzt. Es ist somit unablässig, sich mit dem Betriebssystem auszukennen, nicht nur für das Studium, sondern perspektivisch auch für die Zeit nach dem Studium im Beruf.
Damit dir die ersten Schritte im Studium mit diesem Betriebssystem nicht so schwer fallen, wurde durch die Fachschaft der Informatik der Unix-Crashkurs konzipiert.
In diesem Skript werden Konsolenausgaben und Befehle in einer `dicktengleichen Schrift` geschrieben. Tastenbezeichnungen werden ebenfalls in einer dicktengleichen Schrift umgeben von eckigen Klammern geschrieben, wie zum Beispiel `[Enter]` oder `[q]`, wenn Tasten nacheinander gedrückt werden sollen, dann werden die zu drückenden Tasten direkt nacheinander geschrieben, zum Beispiel `[w][q]`, wenn Tasten gleichzeitig gedrückt werden sollen, dann werden sie mit einem Pluszeichen verknüpft, `[STRG]+[w]`. Werden Alternativen aufgezählt, so werden sie durch ein Komma oder Semikolon voneinander getrennt. Variablenplatzhalter sind ebenfalls in einer dicktengleichen Schrift geschrieben und angeführt von einem Ausrufezeichen in geschweiften Klammern gesetzt, `${Variable}` ist ein Beispiel.

4 Grundlagen

Wie schon in der Einführung erwähnt besitzen Unix-artige Betriebssysteme ein Dateisystem, das eine Baumstruktur darstellt. Dies bedeutet, dass es einen Wurzelknoten gibt, von dem aus sich weitere Knoten abzweigen. In Unix ist alles eine Datei, somit ist auch der Wurzelknoten eine Datei, jedoch handelt es sich hierbei um eine spezielle Datei, nämlich einen sogenannten Ordner. Ein Ordner kann selbst wieder null oder mehr Dateien enthalten, die wiederum Ordner oder Dateien darstellen. Im allgemeinen Sprachgebrauch differenziert man zwischen Ordner und Datei, da ein Ordner Dateien enthält, eine Datei jedoch Daten. Diese Daten können menschenlesbar sein, Binärdaten (die nur von Anwendungen gelesen werden können), es können Anwendungen sein, oder aber sogar Gerätedateien (die die Schnittstellen zur Hardware bilden, auf der das Betriebssystem läuft). Bei der Identifizierung der Dateien und Ordner wird unter Unix-artigen Betriebssystemen die Groß- und Kleinschreibung unterschieden, der Ordner `~/dokumente` ist also etwas anderes als der Ordner `~/Dokumente`, die Datei `Protokoll` ist etwas anderes als die Datei `protokoll`.
Das Betriebssystem selbst benutzt man mit Programmen, die als Schnittstelle zwischen dem Benutzer und dem Betriebssystem fungieren. Hierbei unterscheidet man zwischen grafischen (GUI - Graphical User Interface) und textbasierten (shell) Programmen, letztere werden auch Kommandozeile oder Kommandozeileninterpreter genannt. Zu den wichtigsten grafischen gehört zum Beispiel Gnome oder KDE, die unter Linux am weitesten verbreitete Kommandozeile ist die bash. Im Unix-Crashkurs werden wir die Kommandozeile verwenden, die ihr allerdings unter eurer gewohnten GUI von eurem PC aus mit einem Hilfsprogramm (ssh - dazu später mehr) verwendet. Befehle in der Kommandozeile gibt man an einem sogenannten Prompt ein, die Befehle entsprechen dem folgenden Schema:

```
$ Befehl Argumente
```

Häufig sind den Parametern ein oder zwei Minuszeichen vorangestellt, es können jedoch auch andere oder gar keine Zeichen vor Parametern verwendet werden - das kommt ganz darauf an, wie die einzelnen Programme geschrieben worden sind. Ein Befehl wird mit der `[Enter]`-Taste ausgeführt, hierbei sollte man jedoch äußerst vorsichtig vorgehen und den Befehl zuvor genau überprüfen, da Nachfragen durch ein Programm unüblich sind und die Befehle fast immer direkt ausgeführt werden. Zum Abbruch eines laufenden Prozess kann die Kombination `[STRG]+[c]` verwendet werden.
Nachfolgende Befehle zeigen exemplarisch das Auflisten aller Dateien im Wurzelverzeichnis des angemeldeten Benutzers - dies ist auch das Verzeichnis, in dem Ihr euch normalerweise nach dem anmelden an einem Unix-Rechner befindet - sowie das gleiche Verzeichnis in seiner Präsentation als Baumstruktur. Die Baumstruktur ist hier flach dargestellt, gleiche Ebenen sind gleich weit eingerückt, wenn ein Knoten keine weiteren untergeordneten Knoten besitzt, so nennt man diesen Knoten auch Blatt oder äußeren Knoten.

```
slommage@hu-berlin.de@gruenau3:~> ls
public_html
slommage@hu-berlin.de@gruenau3:~> tree
.
```

```
└── public_html
    ├── screenshot_001.png
    └── screenshot_002.png

1 directory, 2 files
```

Du kannst auf der Kommandozeile die [TAB]-Taste - das ist die mit den zwei waagerechten Pfeilen, die nach links und rechts gegen eine Wand gehen - verwenden, um Befehl oder Dateinamen zu vervollständigen. Um die zuletzt abgesetzten Befehle noch einmal zu wiederholen kannst du die [↑]-Taste verwenden und den Befehl mit der [Enter]-Taste ausführen. Mit der Tastenkombination [STRG]+[r] kannst du in bisher eingegebenen Befehlen suchen. Mit dem Befehl `history` bekommst du eine Historie der von dir bisher eingegebenen Befehle, die jedoch je nach Systemeinstellungen und Kommandozeile nur bis zu einer bestimmten Anzahl oder einem bestimmten Alter zurückgehen. Eine Wiederausführung eines Befehls aus der History kannst du mit dem Befehl `!${Befehlsnummer}` durchführen. Nachfolgend siehst du die Eingabe von `ls pub [TAB][TAB][TAB]`, im Anschluss [↑], gefolgt von [STRG]+[r] [Enter], einer Ausführung von `history` und dann die erneute Ausführung des Befehls 1.

```
slommage@hu-berlin.de@gruenau3:~> ls public_html
public_html/  public_html2
(reverse-i-search)'': touch public_html2
slommage@hu-berlin.de@gruenau3:~> history
1  ls
2  touch public_html2
3  history
slommage@hu-berlin.de@gruenau3:~> !1
ls
public_html  public_html2
slommage@hu-berlin.de@gruenau3:~>
```

Neben den bereits aufgeführten Befehlen gibt es natürlich eine Vielzahl weiterer Befehle für die Systemverwaltung und -betreuung. Diese Befehle sind teilweise in die Kommandozeile eingebaut, zum größten Teil jedoch sind es zusätzliche Programme, die entweder über den Paketmanager der eingesetzten Distribution in das System gelangt sind, vom Anwender aus den Quellen selbst übersetzt worden sind - entweder mit oder ohne Anpassungen des Quelltests - oder aber gar vom Anwender komplett selbst entwickelt worden sind. Ein paar grundlegende dieser Befehle werden wir im Laufe des Unix-Crashkurses behandeln. Außerdem kann über die Kommandozeile auch die Zusammenverwendung von Kommandozeilenbefehlen und Anwendungen verskriptet werden, so dass man mittels eines einzelnen Aufrufs mehrere Dinge erledigen kann. Darauf gehen wir im Kapitel Shellskripting ab Seite 21 näher ein.
Für die Benutzung der Kommandozeile wichtige Tastenkürzel sind in der nachfolgenden Tabelle aufgeführt.

```
[TAB]           Autovervollständigung von Befehlen/Dateinamen/Pfaden
[↓],[↑],[→],[←] Befehle durchblättern oder Cursorposition verändern
```

```
[STRG]+[r]      Befehle durchsuchen
[STRG]+[c]      Aktuellen Prozess beenden (SIGINT)
[STRG]+[z]      Aktuellen Prozess pausieren
[STRG]+[a]      Anfang der Zeile
[STRG]+[e]      Ende der Zeile
[STRG]+[Bild↑]  Seitenweise hochscrollen
[STRG]+[Bild↓]  Seitenweise runterscrollen
```

Je nach Umgebung kann für `[STRG]+[a]` auch `[Pos1]` und für `[STRG]+[e]` auch `[Ende]` verwendet werden oder es funktionieren die Tastenkombinationen für seitenweises scrollen nicht.
Nachdem wir einen kurzen Überblick über die Handhabung der Kommandozeile gegeben haben kommen wir nun zum navigieren in der Kommandozeile. Es wurde schon erwähnt, dass man sich nach dem anmelden an einem Unix-System normalerweise im Nutzerverzeichnis des angemeldeten Nutzers befindet. Dieses Verzeichnis wird auch Homeverzeichnis oder Stammverzeichnis genannt und normalerweise im Prompt mit einer Tilde `~` angezeigt. Mit der Tilde kann man auch von jedem beliebigen Punkt im Verzeichnisbaum aus wieder sein Benutzerverzeichnis referenzieren. Daneben gibt es noch den Schrägstrich `/` als Wurzelverzeichnis des Dateisystems, den Punkt `.` als Referenz auf den Ordner in dem man derzeit steht, sowie zwei Punkte `..` als übergeordneten Ordner zu dem Ordner in dem man derzeit steht. Wenn man eine Datei oder einen Ordner mit einem Punkt als erstem Zeichen benennt, so erhält man eine versteckte Datei oder einen versteckten Ordner.

```
slommage@hu-berlin.de@gruenau1:~/testfolder> ls .
testfile2
slommage@hu-berlin.de@gruenau1:~/testfolder> ls ..
testfile  testfolder
slommage@hu-berlin.de@gruenau1:~/testfolder> ls /
afs  bin  boot  dev  etc  glusterfs  home  lib  lib64  lost+found (...)
```

Der Prompt ist der Teil der Kommandozeile von Anfang der Zeile bis zu der Stelle, ab der du Befehle eingeben kannst. Diesen Prompt kannst du auch an deine Wünsche anpassen.

Einen Befehl um sich den Inhalt des Verzeichnis in dem man sich aktuell befindet anzeigen zu lassen haben wir mit `ls` auch schon weiter oben kennen gelernt. Neben diesem Befehl gibt es noch weitere, zum Beispiel `pwd`, um sich die aktuelle Position im Dateibaum, den sogenannten Pfad angeben zu lassen, sowie den Befehl `cd`, mit dem man von der derzeitigen Ordnerposition in eine anderen Ordner wechseln kann. Hierbei ist anzumerken, dass man nur in Ordner wechseln kann, nicht in (andere) Dateien. Nachfolgender Befehl zeigt den Wechsel in den `/tmp`-Ordner, in dem - wie der Name schon sagt - temporäre Dateien durch das Betriebssystem, Anwendungen oder Nutzer abgelegt werden können.

```
slommage@hu-berlin.de@gruenau3:~> pwd
/home/informatik/slommage
slommage@hu-berlin.de@gruenau3:~> cd /tmp
slommage@hu-berlin.de@gruenau3:/tmp> cd -
/home/informatik/slommage
slommage@hu-berlin.de@gruenau3:~> ls /tmp
ansible_5xyfSi                                  ansible_Yrt1HB (...)
```

In der angegebenen Befehlsfolge sieht man auch gut die Verwendung von Parametern für die Befehle `cd` und `ls`. Der Befehl `cd` wird mit dem Parameter `-` ausgeführt, wodurch man in das Verzeichnis zurückgelangt, aus dem man kam, der Befehl `ls` wurde mit dem Pfad `/tmp` aufgerufen und zeigt somit nicht mehr wie beim Direktaufruf auf Seite 4 den Inhalt des Ordners an, in dem man sich befindet, sondern den Inhalt des Ordners, den man als Parameter übergeben hat. Falls der angegebene Ordner nicht vorhanden ist, wird der Befehl mit einer Fehlermeldung beendet, außerdem beendet sich der Befehl mit einem Fehlercode, auf die wir in einem späteren Kapitel noch eingehen werden.
Neben den Navigationsbefehlen im Dateisystem gibt es auch Befehle um Dateien zu verschieben, kopieren oder zu löschen. Diese kannst du der nachfolgenden Liste entnehmen.

```
mv ${Quelle} ${Ziel}
cp ${Quelle} ${Ziel}
rm ${Ziel}
mkdir ${Ziel}
```

Hierbei ist - wie zuvor schon gewarnt - zu beachten, dass die Befehle ohne Nachfrage ausgeführt werden. Ein mv oder rm führt also dazu, dass das angegebene Ziel - so man Schreibzugriff darauf hat - sofort überschrieben wird. Bei langlaufenden Aktionen kann man, wenn man es bemerkt hat, allerhöchstes das Schlimmste noch verhindern, indem man mittels [STRG]+[c] die Ausführung abbricht. Nichtleere Ordner lassen sich nur löschen, indem man `rm -r` aufruft, dadurch wird der Ordner samt Inhalt rekursiv gelöscht. Möchte man mehr als eine Ordnerebene mit anlegen, so muss man `mkdir -p ${ordnername1}/${ordnername2}` angeben.
Es gibt alleine für die angeführten Befehle eine ganze Menge weiterer Parameter. Wie man diese kennenlernen kann wird im nächsten Kapitel behandelt.

Anwendungen sind nur so gut (und nützlich) wie ihre Dokumentation. Ohne kann der Nutzer der Anwendungen nur raten, was eine Anwendung warum macht, oder wie er eine Anwendung korrekt anwendet. Das haben wir schon mit den bisher als Beispiel aufgeführten

Befehlen gesehen: Versucht man einen Ordner mittels `rm ${Ordnername}` zu löschen der nicht leer ist bekommt man nur eine Fehlermeldung zurück, der Ordner bleibt aber samt Inhalt erhalten. Ohne eine Dokumentation für diesen Befehl kann man sich nicht sicher sein, ob und gegebenenfalls wie man diesen Ordner löschen kann. Hier kommt die Dokumentation ins Spiel. Diese wird im Idealfall durch die Instanz erstellt, die auch die Anwendung entwickelt hat, da nur diese genau weiß, was seine Anwendung wie machen kann. Hierbei kann es sich um eine Einzelperson oder eine Gruppe von Personen handeln. In großen Projekten gibt es sogar eigene Teams, deren Aufgabe es ist die Dokumentation nach den Informationen aus der Entwicklung zu erstellen.
Unter Unix gibt es im allgemeinen zwei Arten der Dokumentation. Zum einen gibt es zu vielen Befehlen einen Parameter, mit dem man eine Anwendung aufrufen kann, der einem eine Kurzübersicht der möglichen Parameter zurückgibt, oftmals mit einer kurzen Beschreibung, was welcher Parameter macht. Dieser Aufrufparameter ist oftmals `-h`, `--help` oder `-?`, es können jedoch auch andere Parameter zum Aufrufen der Hilfe verwendet werden. Ein Aufruf der im Befehl einprogrammierten Hilfe wird nachfolgend am Beispiel von `netstat` gezeigt:

```
slommage@hu-berlin.de@gruenau3:~> netstat -h
usage: netstat [-vWeenNcCF] [<Af>] -r         netstat {-V|--version|-h|--help}
       netstat [-vWnNcaeol] [<Socket> ...]
       netstat { [-vWeenNac] -i | [-cnNe] -M | -s [-6tuw] }

        -r, --route              Routentabelle anzeigen
        -i, --interfaces         Schnittstellentabelle auflisten
        -g, --groups             Mitgliedschaft in Multicastgruppen anzeigen
        -s, --statistics         Netzwerksstatistiken anzeigen (wie SNMP)
(...)
```

Eine spezielle Art der Dokumentation ist es, ein Programm so zu entwickeln, dass es beim aufruf mit einem unbekannten Parameter entweder direkt die möglichen Parameter ausgibt, oder aber den korrekten Parameter ausgibt mit dem man die eingebaute Dokumentation aufruft, wie hier am Beispiel von `cp` gezeigt:

```
slommage@hu-berlin.de@gruenau3:~> cp -N
cp: Ungültige Option -- N
„cp --help" liefert weitere Informationen.
```

Neben dieser einprogrammierten Dokumentation gibt es unter Unix noch die sogenannten Manpages, hierbei handelt es sich um formatierte Softwaredokumentationen, die für verschiedene Aspekte eines Programms erstellt werden können. Die Manpages haben den Vorteil, dass sie eine Offline-Hilfe darstellen, nach der Installation von Manpages eines Programms ist zum Lesen keine Internetverbindung mehr notwendig. Der Aufruf erfolgt im allgemeinen mittels `man ${Befehl}` und mit `apropos ${Suchwort}` kann man innerhalb der installierten Manpages suchen und der Befehl `whatis ${Befehl}` gibt eine Kurzbeschreibung zum Befehl. Je nach verwendeter Distribution kann es jedoch vorkommen, dass die Manpages für Anwendungen entweder nicht, oder nur teilweise installiert worden sind oder dass die Suche mit `apropos` nicht funktioniert. Ein Beispielaufruf einer Manpage wird hier nicht gegeben,

da die Formatierung und Länge das Format dieses Skripts sprengen würde. Dies kannst du jedoch jederzeit auf einem Unix-Rechner deiner Wahl mit zum Beispiel `man man` selbst durchführen, womit du dir die Manpage für die Manpages anschaust. Je nach Umgebung kann es sein, dass der Aufruf von `man` die Wahl lässt, welcher Abschnitt der verfügbaren Dokumentation aufgerufen werden soll.
Innerhalb einer Manpage kann man normalerweise mit den Pfeiltasten `[↓]`,`[↑]`,`[→]`,`[←]` oder Positionstasten `[Bild↑]`,`[Bild↓]`,`[Pos1]`,`[Ende]` navigieren. Das Suchen funktioniert mit der Taste `/${Suchwort}[Enter]`, normalerweise wird dann jedes Vorkommen des Suchworts markiert und man kann mit `[n]` zur nächsten Zeile und mit `[Shift]+[n]` zur vorherigen Zeile, in der das Suchwort vorkommt, springen. Die Anzeige einer Manpage kann jederzeit mit der Taste `[q]` beendet werden.

5 Anwendungen für Dateioperationen

Nachdem wir in den vorherigen Kaptieln in den Grundlagen schon einige Dateibefehle wie `cp`, `mv` oder `rm` kennen gelernt haben, sowie Mittel, mit denen wir nähere Informationen zur Ausführung von Programmen herausfinden können folgen hier noch ein paar weitere wichtige Befehle, mit denen man sich Inhalte von Dateien anzeigen lassen kann oder mit denen man die Inhalte von Dateien ändern kann.
Der Befehl `less ${Datei}` kann verwendet werden, um den Inhalt von Dateien zeilenweise anzuzeigen. Da dieses Programm normalerweise als Anzeigeprogramm für Manpages verwendet wird, funktionieren während man less verwendet alle Tastenkombinationen, die schon im vorhergehenden Kapitel unter dem Punkt Manpages auf Seite 9 beschrieben worden sind.
Möchte man Inhalte in Dateien finden, so bietet sich das Programm
`grep ${Suchwort} ${Datei}` an. Hierbei werden alle Zeilen in denen das Suchwort in der übergebenen Datei auftaucht ausgegeben. Suchwort muss hier kein echtes Wort sein, sondern bezeichnet hier eine beliebige Zeichenkette. Hat man mehrere Dateien spezifiziert, so wird vor jedem Vorkommen die Datei mit angegeben. Dateien können einzeln angegeben werden oder aber mittels Globbing ausgewählt werden. Mehr zu Globbing erfährst du im gleichnamigen Kapitel ab Seite **??**. Häufig werden die Fundstellen der Suchworte farbig hervorgehoben, damit man sie leichter in der Ausgabe findet, wenn es die verwendete Umgebung unterstützt. Nachfolgend noch ein Aufruf mit mehreren Dateien die durchsucht werden sollen und Angabe der Zeile (nach dem ersten Doppelpunkt) in der das Suchwort auftaucht. ;tabsize

```
slommage@hu-berlin.de@gruenau3:~> grep -n 4 testcases/*small.stdin
testcases/example_fully_connected_small.stdin:1:d4:lm,da
testcases/example_fully_connected_small.stdin:2:31:d4,lm,da-608
testcases/example_ring_small.stdin:2:ha:uj,w4-320
testcases/example_ring_small.stdin:3:kn:w4-474
```

Häufig sind `grep`-Varianten mit erweitertem Funktionsumfang erhältlich, die dann mit einem anderen Namen aufgerufen werden, damit der originale Aufruf von `grep` für Kommandozeilenskripting kompatibel bleibt. Erwähnenswert ist hier zum Beispiel `egrep`, welches mit

sogenannten Regex - also regulären Ausdrücken - umgehen kann. Was ein regulärer Ausdruck ist wirst du im Studium im Modul „Einführung in die theoretische Informatik“ noch näher kennenlernen. Zu Beachten ist noch, dass `grep` und davon abgeleiteten Programme keine Suchwörter in Binärdateien finden können.
Wenn man Dateiinhalte sortieren möchte bietet sich der Befehl `sort ${Datei}` an, der über Parameter verfügt, mit dem man die Spaltendelimiter spezifizieren kann, sowie die Möglichkeit Sortierreihenfolgen und Sortierschlüssel festzulegen.
Ein weiterer wichtiger Befehl ist `cat ${Datei}`, mit dem ein oder mehrere Dateien aneinandergehängt und auf die Standardausgabe geschrieben werden können. Mehr zur Standardausgabe findest du im Kapitel stdio ab Seite **??**. Wie man die Ausgabe von und die Eingabe für nicht nur von den hier angeführten Programmen geeignet verknüpft findest du ebenfalls in diesem Kapitel.
Um Dateien zu packen oder entpacken kann der Befehl `tar` verwendet werden, für den es verschiedene Aufrufparameter gibt um das Verhalten zu beeinflussen. Eine durch `tar` erstellte Datei wird Archiv genannt, der Name ist eine Abkürzung für „tape archive“ dar. Die wichtigsten sind die nachfolgende aufgeführt:

```
-f  Archiv das verwendet wird
-t  Inhalt auflisten
-u  Dateien die neuer sind einfügen
-c  Archiv estellen
-x  Dateien aus Archiv extrahieren
-r  Dateien an Archiv anhängen
```

Als letzten Punkt in diesem Kapitel gehen wir jetzt noch auf zwei Dateieditoren ein.
Zum einen ist gibt das Programm Nano. Mit `nano ${Datei}` startet man den Editor, wenn die angegebene Datei noch nicht existiert wird sie vorerst im Arbeitsspeicher des Rechners bearbeitet und beim Verlassen des Editors auf Nachfrage mit dem im Editor verfassten Inhalt gespeichert. Falls die Datei schon existiert wird sie mit ihrem Inhalt zum bearbeiten geöffnet. Die wichtigsten Tastenkombinationen sind in den zwei Fußzeilen des Editors angeführt, hierbei ist mit ^ die Taste `[STRG]` gemeint.
Das andere - und am häufigsten vorinstallierte - Programm ist der Texteditor `vi ${Datei}`, die Version mit erweitertem Funktionsumfang `vim ${Datei}` oder das nur zum betrachten von Dateien gedachte `view ${Datei}`. Häufig ist, auch aus Kompatibilitätsgründen, ersteres ein Link auf das Zweite. Die Bedienung von vi ist etwas ungewohnter als die von Nano, vor allem, da es hier keine leicht sichtbare Hilfe für Tastenkombinationen gibt. Es gibt bei der Benutzung von vi zwei Benutzungsmodi, standardmäßig startet man vi im Befehlsmodus, in dem zum navigieren und blättern die gleichen Tastenkombinationen verwendet werden können wie in einer Manpage oder dem Programm `less`. Außerdem kann man hier mit verschiedenen Tastenkombinationen den Text verändern, ein paar dieser Tastenkombinationen kannst du dir in einem sogenannten vi-CheatSheet anschauen. [8] Mit der Taste `[i]` (einfügen unter dem Cursor) oder `[a]` (einfügen nach dem Cursor) kommt man in den Einfügemodus, in dem man den Dateiinhalt zeichenweise verändern kann. Diesen Modus kannst du mit der Taste `[Esc]` wieder verlassen. Um vi ganz zu benden drückst du im Befehlsmodus die Taste `[:]` gefolgt von der Tastenkombination `[q][Enter]`, solltest du Änderungen durchgeführt haben, die du nicht anbringen möchtest, so musst du direkt nach dem `[q]`

noch ein [!] eingeben. Falls du die Änderungen anbringen möchtest, so musst du die Tastenkombination [w][q][Enter] eingeben. Es gibt noch viele weitere Befehle um Worte, von Cursor bis Zeilenende oder Zeilenanfang zu löschen oder zu kopieren, wie im vi-Cheat-Sheet ersichtlich. Wenn man sich nicht sicher ist, in welchem Modus man sich gerade befindet hilft es mehrfach [Esc] zu drücken, um garantiert in den Befehlsmodus zurückzukehren. Ist man GUI-Texteditoren wie zum Beispiel Microsoft Word oder LibreOffice Writer gewöhnt kann es schnell passieren, dass man ausversehen [Strg]+[s] drückt, weil man damit speichern möchte, dies aktiviert in der Kommandozeile einen sogenannten Freeze lock, man sieht dann die Eingaben nicht mehr. Diesen Modus kann man durch [Strg]+[q] wieder verlassen, dann werden auch alle in der Zwischenzeit getätigten Eingaben angezeigt. Dies alles führt dazu, dass man oft scherzhaft ein gestartetes vi als Zufallsgenerator für damit unerfahrene bezeichnet.

6 Benutzer

Damit nicht jeder angemeldete Nutzer auf einem Unix-System alles machen kann verfügt Unix über ein Rechtemodell. Hierbei gibt es einen Benutzer, der alles darf - unter Unix root genannt und der die User-ID (UID) 0 besitzt. Jeder Nutzer gehört außerdem mindestens einer Gruppe an, der Nutzer root ist hier in der Gruppe root mit der Gruppen-ID (GID) 0. Häufig ist die Standardgruppe eines Nutzers gleich seinem Nutzernamen. Alle weiteren Nutzer verfügen nur über eingeschränkte Rechte. Normalerweise haben manuell eingerichtete oder menschliche Nutzer eine UID und GID ab 1000, die Nummern zwischen 0 und 999 sind für Systemdienste und -anwendungen reserviert. Falls man für irgendetwas Rechte benötigt, über die der eigene Nutzer nicht verfügt, kann man entweder den Nutzer wechseln, oder man verwendet das Programm sudo. Häufig sind die weiteren Nutzerkennungen mit einem Kennwort abgesichert, für sudo kann feingranular eingestellt werden, was der aufrufende Nutzer alles aufrufen darf. Standardmäßig sind alle im System bekannten Nutzer in der Datei /etc/passwd abgespeichert, alle im System bekannten Gruppen in der Datei /etc/group. Passwortinformationen werden standardmäßig in der Datei /etc/shadow gespeichert, unter anderem ist hier auch das Passwort verschlüsselt abgelegt. Beispieleinträge für alle drei Dateien siehst du nachfolgend:

```
# head -1 /etc/passwd /etc/shadow /etc/group
==> /etc/passwd <==
root:x:0:0:root:/root:/bin/bash

==> /etc/shadow <==
root:$6$AX1sug.SUMlGhKAM$OzrzjRGisbgcX5bDC8Xlmz.yVrViCNLTLPmF3df11t.AODyvOwRgOn12E8J

==> /etc/group <==
root:x:0:
```

In der /etc/passwd ist neben dem Benutzername mit dem x in der zweiten Spalte angezeigt, dass ein Kennwort für den Nutzer in der /etc/shadow abgelegt ist. Die weiteren Spalten sind UID und GID, gefolgt von einem Klartext-/Anzeigename. Im Anschluss ist dann noch das

Benutzerverzeichnis angegeben, hier `/root`, sowie die Standardkommandozeile die gestartet wird, wenn sich der Benutzer anmeldet, in diesem Fall `/bin/bash`. Zur Verwaltung von Kennwörtern stehen die nachfolgenden Befehle zur Verfügung.

```
useradd     hinzufügen von Nutzern
userdel     entfernen von Nutzern
groupadd    hinzufügen von Gruppen
groupdel    entfernen von Gruppen
passwd      Kennwortänderung von Nutzern
chage       Änderung der Passwortalterung
```

Diese Dateien sind jedoch für dich für die Arbeit auf den Informatik-Servern nicht von belang, da auf diesen Servern die Authentifizierung via ldap erfolgt.
Wie schon angeführt hat jeder Nutzer gewisse Rechte. Außerdem gehört jeder Prozess einem Nutzer. Die Rechte eines Nutzers werden über Informationen an den Dateien und Ordnern festgelegt. Man kann sich die Rechte einer Datei mit dem Befehl `ls -l` anzeigen lassen:

```
slommag@gruenau3:~> ls -l
insgesamt 2048
drwxr-xr-x 2 slommage mi20 0 14. Jan 2021  public_html
-rwxr--r-- 1 slommage mi20 0 28. Sep 21:41 public_html2
drwxrwxrwx 2 slommage mi20 0 29. Sep 23:18 testcases
drwxr-xr-x 2 slommage mi20 0 29. Sep 14:52 testdir0
-rw-rw-rw- 1 slommage mi20 8 29. Sep 22:07 test.out
lrwxrwxrwx 1 slommage mi20 8 29. Sep 22:50 test.out.lnk -> test.out
```

Die zehn ersten Spalten der Ausgabe sind die sogenannten Dateiflags, Berechtigungsflags oder Rechteflags, anstelle von Flag kann man hier auch Bit sagen. Hier sieht man gut, dass im Nutzerverzeichnis Dateien und Ordner liegen, in der ersten Spalte kann man anhand des Flags `d` die Ordner erkennen, das Flag `l` zeigt einen Link - eine Dateiverknüpfung - an. Die neun Nachfolgenden Spalten sind jeweils 3er Gruppen, die man auch als Bitmaske auffassen kann, und bedeuten jeweils `rwx` und geben die Rechte für diese Datei oder den Ordner für den Nutzer (hier slommage), die Gruppe (hier mi20) sowie alle anderen Nutzer an. Die einzelnen Rechte sind: Leserechte (`r`), Schreibrechte (`w`), Ausführungsrechte (`x`). Wie schon im Kapitel Grundlagen auf Seite 4 erklärt ist ein Ordner eine spezielle Datei, mit dem Flag `x` zeigt man an, dass das Recht gesetzt ist, in diesen Ordner wechseln zu dürfen. Ein `-` signalisiert, dass ein Flag nicht gesetzt ist. Für das setzen von Berechtigungen gibt es die Befehle `chmod` zum setzen der Flags, `chown` um die Datei oder den Ordner Nutzern (und Gruppen) zuzuordnen, sowie `chgrp` um die Gruppezuordnung zu ändern. Die Änderung von Rechten kann wie nachfolgend beschrieben erfolgen.

```
gruenau1 slommage ( testfolder ) $ ls -l testfile
-rw------- 1 slommage mi20 0  1. Okt 13:56 testfile
gruenau1 slommage ( testfolder ) $ chmod 700 testfile; chmod o+x testfile
gruenau1 slommage ( testfolder ) $ ls -l testfile
-rwx-----x 1 slommage mi20 0  1. Okt 13:56 testfile
```

Man kann gut sehen, wie durch die Bitmaske `700` die Flags `rwx------` gesetzt worden sind $(1 * 2^2 + 1 * 2^1 + 1 * 2^0)$, im Anschluss wurde dann noch das Ausführungsflag für alle anderen gesetzt. Für die Verwendung der Bitmaske müsst ihr das Binärsystem kennen, wenn ihr damit noch nicht vertraut seid, das wird spätestens in „Grundlagen der Programmierung“ behandelt.
Neben den drei angeführten Standardflags gibt es auch noch Spezialflags, nämlich das SUID-Flag (Set UID), ist dieses gesetzt, so steht an der Stelle der Nutzerausführungsberechtigung ein `s` anstellte des dort gesetzten `x` und ein großes `S`, falls das `x` für den Nutzer nicht gesetzt ist. Dieses Flag bewirkt, dass ein Nutzer, der ein mit diesem Recht versehene Programm ausführt für die Zeit der Ausführung die Nutzer-ID des Programmeigentümers erhält - es handelt sich hierbei also um ein ziemlich mächtiges und auf gefährliches Flag. Ein weiteres Spezialflag ist das SGID-Flag (Set GID), ist dieses Flag an einem Ordner gesetzt, man erkennt es an einem `s` an der Stelle der Gruppenausführungsrechtigung, so gehören alle Dateien und Ordner, die in der Hierarchie unterhalb dieses Ordners erstellt werden der Gruppe die dieser Ordner hat. Dies ist nützlich, wenn ein Systemverwalter Ordner erstellt, die von anderen Nutzern weiterverwendet werden.

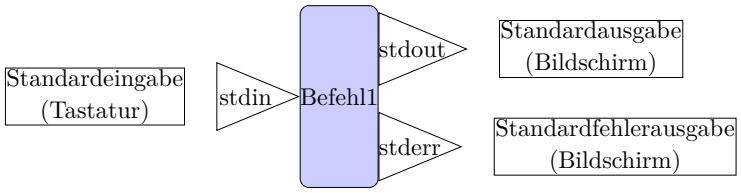
```
[root@centos tmp]# mkdir -p folder/folder1 folder/folder2
[root@centos tmp]# chgrp testuser folder/folder1 folder/folder2
[root@centos tmp]# chmod g+s testfolder/testfolder1
[root@centos tmp]# touch folder/folder1/file1 folder/folder2/file2
[root@centos tmp]# tree -pug folder/
folder/
├── [drwxr-sr-x root     testuser]  folder1
│   └── [-rw-r--r-- root     testuser]  file1
└── [drwxr-xr-x root     testuser]  folder2
    └── [-rw-r--r-- root     root    ]  file2
```

Das letzte Spezialflag ist das Sticky-Flag, dieses wird durch ein `T` anstelle der Ausführungsberechtigung für alle anderen angezeigt und verhindert, dass andere als der Eigentümer (ausgenommen root) diesen Ordner oder untergeordnete Dateien oder Ordner löschen können.
Mit den Befehlen `whoami` oder `id` könnt Ihr herausfinden, wie der Benutzer heißt, mit dem ihr angemeldet seid und in welcher Gruppe Ihr euch befindet. Welche Nutzer alles derzeit an einem System angemeldet sind kann man mithilfe des Befehls `who` erfahren.
Neben diesen rudimentären Berechtigungen auf Nutzer:Gruppe:Andere-Ebene gibt es noch die Linux Access Control Lists (ACL), mit der für jede Datei und jeden Ordner feingranular der Zugriff mit den gleichen drei `rwx`-Flags gesteuert werden kann. Dies ist aber ein Thema der tiefergehenden Systemverwaltung und soll hier nicht weiter erläutert werden.

7 stdio

Damit Anwendungen sowohl Eingaben verarbeiten, als auch Daten ausgeben können gibt es unter Unix die Standardbibliothek `stdio`. Diese kapselt die Ein- und Ausgaben als ge-

pufferten Datenstrom von den dahinter liegenden physischen Ein- und Ausgaben. Es gibt einen Eingabepfad, `stdin`, und zwei Ausgabepfade, `stdout` sowie `stderr`. Standardmäßig wird als `stdin` die Tastatur verwendet und als `stdout` und `stderr` die Kommandozeile (also der Bildschirm).



Die Ein- und Ausgabeströme kann man auch miteinander verknüpfen, indem man zum Beispiel die `stdout` eines Programms an die `stdin` eines weiteren Programms übergibt. Diesen Vorgang nennt man „pipen“, das dazu benötigte Zeichen ist auch die Pipe `[|]`, du findest sie auf einer PC-Tastatur rechts neben der `[Shift]`-Taste und benötigst die `[Alt Gr]`-Taste zum aufrufen, unter MacOS ruft man sie über `[Alt]+[7]` auf.. Hier im Beispiel sieht du ein `tail`, also ein permanenten Aufruf der letzten Zeile einer Datei, gepiped in `grep root`, also einer Suche nach root.

```
[root@centos ~]# tail -f /var/log/secure|grep root
Oct  1 22:20:50 centos su[124053]: pam_unix(su-l:session): session (...)
Oct  1 22:21:12 centos su[124190]: pam_unix(su-l:session): session (...)
```

Neben dem Pipe-Zeichen wird auch das Kleiner-Zeichen `<` verwendet, um Daten oder Dateien in die Standardeingabe umzuleiten, sowie das Größer-Zeichen `>`, um die Standardausgabe an Dateien umzuleiten, wobei der Dateiinhalt überschrieben wird, falls die Datei schon vorhanden ist. Möchte man die umgelenkten Daten an eine bereits vorhandene Datei anfügen, so muss man das Größer-Zeichen doppelt angeben `>>`. Der Ausgabestrom `stdout` wird auch mit einer `1` referenziert, der Ausgabestrom `stderr` mit einer `2`. Nachfolgend Beispiele für die Umlenkung jeweils nach `/dev/null`, sowie die zusammenführung beider Ausgabeströme.

```
slommage@hu-berlin.de@gruenau1:~> cat /etc/hostname 1>/dev/null
slommage@hu-berlin.de@gruenau1:~> cat /etc/hostname 2>/dev/null
gruenau1.informatik.hu-berlin.de
slommage@hu-berlin.de@gruenau1:~> cat /etc/hostname >/dev/null 2>&1
```

Mit einer Verkettung einzelner Befehle kann man somit mächtige Operationen erstellen und sich Daten die man benötigt aus anderen Datenquellen zusammensammeln und in eine neue Datenquelle überführen. Nachfolgend noch ein paar Programme, die dafür gut verwendet werden können:

```
wc      Zählen von Bytes, Zeichen, Zeichenketten, Zeilen
tee     Ausgabe nach stdout und Datei
head    Ausgabe der führenden Zeilen einer Datei
tail    Ausgabe der letzten Zeilen einer Datei
```

Nachfolgender Befehl führt eine Anwendung aus und vergleicht den Output mit einem vorliegenden, unsortierten Beispieloutput. Das `echo $?` gibt hier den Returncode der letzten

Anweisung aus, mithilfe der Manpage kann man erfahren, dass die beiden verglichenen Dateien gleich sind. Das -Zeichen als zweiter Parameter bei `diff` referenziert auf den reingepipten Input - natürlich hätte man hier auch alles vor dem zweiten Pipe-Zeichen ebenfalls in `<()` setzen können und auf das zweite Pipe-Zeichen verzichten können, die zweite Zeile ist identisch zur ersten.

```
[test@centos ~]# ./prog <xmpl.in|sort|diff -u - <(sort ./xmpl.out) && echo $?
0
[test@centos ~]# diff -u <(./prog <xmpl.in|sort) <(sort ./xmpl.out) && echo $?
0
```

8 Globbing

Nachdem wir in den vorhergehenden Kapiteln gesehen haben, wie man Operationen auf Dateien durchführen kann stellt sich die Frage, wie man mehrere Dateien auswählen kann. Hierfür steht das sogenannte globbing zur Verfügung. Dateinamen folgen einem bestimmten Muster, man kann mittels globbing mit Platzhaltern, sogenannten Wildcards, diese Dateimuster expandieren. Diese Expandierung wird durch den Kommandozeileninterpreter - also in der Shell - durchgeführt und findet vor der Ausführung des Befehls statt, mit dem man die ausgewählten Dateien verwenden möchte. [9] Die Folgende Übersicht gibt ein paar Globbing-Schemen an.

```
*           Beliebiges Zeichen, 0 bis beliebig oft
?           Beliebiges Zeichen genau 1 mal
{abc,123}   Zeichenkette, alternativen kommagetrennt
[a-f]       Zeichen aus dem gewählten Bereich, Alternativen kommagetrennt
```

Hier ein Beispielt für die Verwendung, an der man gut sehen kann, dass man genau aufpassen muss, was man warum auswählt.

```
gruenau1 slommage 1 (~/testfolder) # ls [ab,cd].pdf
ls: Zugriff auf '[ab,cd].pdf' nicht möglich: Datei oder Verzeichnis nicht(...)
gruenau1 slommage 2 (~/testfolder) # ls {ab,cd}.pdf
ab.pdf  cd.pdf
gruenau1 slommage 3 (~/testfolder) # ls [a-c][b-d].pdf
ab.pdf  cd.pdf
gruenau1 slommage 4 (~/testfolder) # ls {a-c}{b-d}.pdf
ls: Zugriff auf '{a-c}{b-d}.pdf' nicht möglich: Datei oder Verzeichnis(...)
gruenau1 slommage 5 (~/testfolder) # ls {ab,cd}*.pdf
ab.pdf  cd.pdf
gruenau1 slommage 6 (~/testfolder) # ls [ab,cd]*.pdf
ab.pdf  cd.pdf
```

Der erste Befehl wählt den Zeichenbereich `a` bis `b` oder `c` bis `d`, gefolgt von `.pdf`.
Der zweite Befehl wählt `ab.pdf` oder `bc.pdf`.

Der dritte Befehl wählt den Zeichenbereich `a` bis `c` gefolgt vom Zeichenbereich `b` bis `d`, gefolgt von `.pdf`.
Der vierte Befehl wählt die Datei `{a-c}{b-d}.pdf`.
Der fünfte Befehl wählt die Zeichenkette `ab` oder die Zeichenkette `cd`, gefolgt von beliebigen Zeichen und gefolgt von `.pdf`.
Der sechste Befehle wählt den Zeichenbereich `a` bis `b` oder `c` bis `d`, gefolgt von beliebigen Zeichen und gefolgt von `.pdf`.
Neben dem Globbing gibt es auch noch reguläre Ausdrücke (regex oder regular expressions), die ein Textmuster beschreiben. Damit formuliert man eine Zeichenkette, die eine andere Zeichenkette beschreibt. Eine Zeichenkette „matcht“ einen regex oder nicht. Man kann reguläre Ausdrücke zur Inputvalidierung verwenden, zur Textsuche - ein Beispiel wurde in diesem Skript schon mit `egrep` angeführt - und sie werden in der theoretischen Informatik verwendet um reguläre Sprachen zu beschreiben. Ein Punkt `.` steht für ein beliebiges Zeichen (Literal). Ein Stern `*` bedeutet, das vorhergehende Zeichen beliebig oft, auch 0 mal, wohingegen ein Plus `+` bedeutet, dass das vorhergehende Zeichen mindestens einmal auftreten muss, aber beliebig oft auftreten kann. Möchte man Zeichen aus einem Bereich auswählen, so setzt man diese in eckige Klammern `[]`, nachfolgend in geschweiften Klammern `{}` kann man die Anzahl an Wiederholungen angeben. Zeichenketten werden in runden Klammern `()` angegeben und Alternativen durch eine Pipe `|` getrennt. Es gibt verschieden mächtige Implementierungen um reguläre Ausdrücke zu schreiben. Die Webseite regex101.com [10] kann man gut verwenden, um reguläre Ausdrücke in verschiedenen Implementierungen gegen Texte zu matchen, dort gibt es auch eine Erklärung, welche Zeichen was bedeuten. Dabei muss man allerdings darauf achten, dass man dort die Satzgrenzen mit `^` und `$` angeben muss. Nachfolgend noch ein paar Beispiele für reguläre Ausdrücke.

```
a+          mindestens ein a
            z.B. {a;aa;aaaaa}
a+b         mindestens ein a gefolgt von genau einem b
            z.B. {ab;aab,aaaaab}
.*ab.*      beliebige Zeichen, dann ab, dann beliebige Zeichen
            z.B. {ab;yxabz;9ab12}
[A-Z]{2}    Zeichenbereich von A bis Z, zweimal hintereinander
            z.B. {AA;YZ;FG}
```

Alle Operationen auf Zeichen machen sich zunutze, dass die Zeichen im verwendeten Zeichensatz alphabetisch angeordnet sind, ein `a` ist somit „kleiner“ als ein `b`, genaueres zu Ordnungen werdet ihr in der Vorlesung „Einführung in die theoretische Informatik“ lernen.

9 ssh

In den bisherigen Kapiteln sind wir davon ausgegangen, dass wir mit einem Unix-System verbunden sind und die Kommandozeile geöffnet haben. In diesem Kapitel werden wir mehr darüber erfahren, wie diese Verbindung zum Server zustande kommt. Das es sich um eine Netzwerkverbindung handelt ist nicht weiter überraschend, solche Verbindungen können jedoch im klartext stattfinden, oder aber verschlüsselt. Ein Beispiel für eine solche Verbindung

wäre eine pure Verbindung mittels eines Telnet-Clients der gegen einen Telnet-Server. Da über solch eine Verbindung natürlich auch die Kennwörter für den Nutzer am Server mit dem man sich verbinden möchte im Klartext übertragen wird stellt eine solche Verbindung ein großes Sicherheitsrisiko dar. Aus eben diesem Grund wird bei Windows oder in einigen Linux-Distributionen der Telnet-Client `telnet` nicht mehr standardmäßig mitinstalliert.
Hier setzt ssh an, bei dem ein verschlüsseltes Protokoll in einer Server-Client-Architektur verwendet wird. Die Verschlüsselung kann mit verschiedenen Algorithmen umgesetzt werden, jedoch müssen Client und Server die selben Algorithmen verstehen. Hautpsächlich eingesetzt werden derzeit `RSA` - eine asymmetrischen Verschlüsselung die auf einem Schlüsselpaar aus öffentlichem und privatem Schlüssel aufbaut und auf großen Primzahlen basiert - mit $\geq$ 2048 bit Schlüssellänge, sowie einer Verschlüsselung, die auf elliptischen Kurven aufbaut - hier wird häufig `ed25519` oder eine NIST-Kurve verwendet - und die ebenfalls eine asymmetrische Verschlüsselung ist. Alte `DSA`-Schlüssel sollen aus Sicherheitsgründen nicht mehr verwendet werden [11], viele aktuelle Implementierungen unterstützen diese auch gar nicht mehr. Mehr zur Funktionsweise von `RSA` werdet Ihr bereits in der Vorlesung „Lineare Arithmetik“ erfahren. Elliptische Kurven werden eventuell im Kurs „IT-Sicherheit“ behandelt.
Es gibt verschiedene Implementationen von ssh, am verbreitetsten ist `openssh`, das sowohl unter Windows 10 zur Verfügung steht, als auch unter nahezu allen Linux-Distributionen.
Bei einem Verbindungsaufbau wird schon die initiale Verbindung verschlüsselt hergestellt, der Server präsentiert einen Fingerprint seines öffentlichen Schlüssel und der ssh-Client gleicht den mit seiner Liste bekannter Schlüssel ab. Ist der Schlüssel nicht bekannt oder passt nicht zum bekannten, so wird dem Nutzer der Fingerprint präsentiert und dieser muss entscheiden, ob er die Verbindung aufbauen möchte. Dies ist ein Sicherheitsmechanismus, damit sich niemand unerkannt als der Server ausgeben kann, mit dem man sich verbinden möchte - es baut aber darauf, dass man auf einem weiteren sicheren Kanal den Fingerprint des Servers erhält, mit dem man sich verbinden möchte. Wurde die sichere Verbindung so erst einmal aufgebaut, dann authentifiziert sich im Anschluss der Nutzer am Server, dies kann entweder mit einem Kennwort geschehen, oder aber ebenfalls über ein Schlüsselpaar, wobei der Fingerprint des öffentlichen Schlüssel des Nutzers normalerweise im Home-Verzeichnis unter `.ssh/id_${gewählter Name}.pub`, `.ssh/authorized_keys` oder `.ssh/authorized_keys2` abgespeichert ist.

```
gruenau1 slommage 1 ( ~ ) $ cat .ssh/authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDZszw(...)jCDW23spgyb8tLDMXF4 slommage
```

Genaueres zu ssh könnt ihr im zugehörigen `RFC 4251` nachlesen. [12]
Neben dem hier schon angeführten ssh-Client zum Verbinden mit dem ssh-Server gibt es noch weitere Tools für die ssh-Infrastrukur, die man der nachfolgenden Übersicht entnehmen kann.

```
ssh          ssh-Client zum Verbinden mit dem ssh-Server
sshd         ssh-Server/Daemon
ssh-agent    Authentifizierungsagent, für leises Schlüsselhandling
ssh-add      um einen Schlüssel dem Agenten hinzuzufügen
ssh-keygen   zur Schlüsselerstellung, Schlüsselprüfung
sftp         verschlüsseltes Dateiübertragungsprogramm, wie ftp
```

```
scp         verschlüsseltes Dateiübertragungsprogramm
rsync       verschlüsselter Dateiübertragungsprogramm mit Abgleich
```

Der weiter oben angeführt Verbindungsaufbau findet mit `ssh ${Username}@${Hostname}` statt, standardmäßig wird hierbei versucht die den Server am Port 22 zu kontaktieren, über den Parameter `-p` kann jedoch auch ein anderer Port gewählt werden auf den der Server hört. Möchte man die Verbindung beendent, so geschieht dies über die Eingabe von `exit`, falls das mal nicht funktionieren sollte, so gibt es noch die Notfalltastenkombination `[Enter][~][.]`. WICHTIG: Allen asymmetrischen Schlüsselverfahren gemein ist, dass der private Schlüssel - mit dem ihr an euch gesandte Nachrichten entschlüsselt - in eurem Besitz und somit geheim bleiben muss. Ihr könnt und solltet für den privaten Schlüssel ein Kennwort vergeben, damit hab ihr sofort die Sicherheits des Schlüssels neben dem Faktor Besitz um den Faktor Wissen erweitert. Der öffentliche Schlüssel ist der, den ihr euren Kommunikationspartnern - im Fall von ssh dem Server - aushändigt, mit diesem werden Nachrichten an euch verschlüsselt.

10 Prozessverwaltung

Nachdem wir in diesem Dokument schon einige Anwendungen eingeführt haben soll es in diesem Kapitel um die Prozessverwaltung gehen. Theoretisch erfolgt die Prozessverwaltung durch das Betriebssystem, im weiteren Sinne kann man jedoch auch die manuellen Steuerung von Prozessen durch einen Benutzer als Prozessverwaltung ansehen. Auf einem Multinutzersystem oder einem Server sind beide Arten der Prozessverwaltung wichtig, um Engpässe vermeiden, erkennen, umgehen oder beheben zu können. Während dem Betriebssystem hierbei der sogenannte Scheduler zur Verfügung steht, um Prozesse auf verfügbare Ressourcen zu verteilen. Außerdem gibt es noch den sogenannten OOM-Reaper (Out Of Memory), der die Anwendung mit der größten Arbeitsspeicherbelegung beendet wenn kein Arbeitsspeicher (RAM) mehr zur Verfügung steht, um wieder Arbeitsspeicher frei zu bekommen. Außerdem kann man noch - zum Beispiel Betriebssystemweit mittels `/etc/security/limits.conf` - für verschiedene Nutzer und Gruppen Ressourcenlimits festlegen, deren Einhaltung vom Betriebssystem überwacht werden.
Für die manuelle Steuerung und Überwachung stehen mehrere Befehle zur Verfügung, zum Beispiel die Anwendungen `ps` oder `pstree`, mit denen man sich einen Überblick über derzeit laufende Prozesse verschaffen kann.

```
slommage@hu-berlin.de@gruenau4:~> ps -ef|grep slommage
root     2257235    2225  0 09:02 ?        00:00:00 sshd: slommage@hu-berlin.de [pri
slommag+ 2260150 2257235  0 09:02 ?        00:00:00 sshd: slommage@hu-berlin.de@pts/
slommag+ 2280377 2260251  0 09:03 pts/3    00:00:00 grep --color=auto slommage
slommage@hu-berlin.de@gruenau4:~> pstree -p 2225
sshd(2225)─┬─sshd(471123)───sshd(490935)───bash(491004)
           ├─sshd(735559)───sshd(735684)───sftp-server(735710)
           ├─sshd(1093346)───sshd(1094346)───sftp-server(1094381)
           ├─sshd(1314594)───sshd(1322693)───bash(1322742)
           ├─sshd(2257235)───sshd(2260150)───bash(2260251)───pstree(2493919)
           └─sshd(2493828)───sshd(2493834)
```

Man kann in dem Beispiel gut sehen, wie ein Prozess durch einen anderen aufgerufen worden ist: Die Prozessid `PID` in der zweiten Spalte - in der die `PID` des Elternprozess angezeigt wird - des ersten Prozess entspricht der `PID` in der dritten Spalte des zweiten Prozess. Dieser Prozess gehört auch noch dem root-User, hat also erhöhte Rechte, darf zum Beispiel ohne weiteres den Standardport von `ssh` öffnen. Der Prozess mit der `PID` 2260150 hingegen gehört dann dem angemeldeten Nutzer, er wurde geforked. Was genau geforked bedeutet sieht man anhand des zweiten Befehls, dort wird ähnlich wie beim Dateibefehl `tree` eine Baumstruktur der laufenden Prozesse angegeben, hier ab dem `sshd`-Prozess - man sieht gut, wie sich die einzelnen Anmeldeprozesse für Nutzer abgabeln - also forken. Die Anwendungen `top` und `htop` zeigen eine konfigurierbare und sortierbare Listenansicht aller derzeit laufenden Prozesse an, standardmäßig nach der CPU-Auslastung sortiert.

```
    top - 09:22:53 up 1 day, 23:52,  2 users,  load average: 26,90, 36,67, 39,25
Tasks: 830 total,   2 running, 828 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,1 us,  0,3 sy,  1,9 ni, 96,4 id,  1,4 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem : 772638,9+total, 633808,2+free, 13351,10+used, 125479,5+buff/cache
MiB Swap: 32767,99+total, 32767,99+free,    0,000 used. 753520,5+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND          nT
27868 chmieles  35  15 4888928  57640   2748 R 209,9 0,007   3259:16 tree_depth_tabu
21349 sddm      20   0 3906180 223392 115632 S 3,960 0,028 144:58.79 sddm-greeter
20312 slommage  20   0   58656   5400   3940 R 1,980 0,001   0:00.31 top
21240 root      20   0 3085748  80496  44508 S 0,990 0,010  11:01.53 X
    1 root      20   0  239996  14528  10268 S 0,000 0,002   2:25.28 systemd
    2 root      20   0       0      0      0 S 0,000 0,000   0:12.21 kthreadd
    (...)
```

Mithilfe der Taste `[f]` kann man die Ansicht auch konfigurieren, mit der Taste `[1]` schaltet mein eine Anzeige der CPU-Kernauslastung an, die bei vielen CPU-Kernen aber je nach Konsolengröße unübersichtlich sein kann. Wichtige Tastenkürzel zur Sortierung sind noch `[Shift]+[P]` für die CPU-Auslastung, `[Shift]+[M]` für die Speicherauslastung und `[Shift]+[T]` für die akkumulierte CPU-Zeit. Alle angezeigten Parameter kann man heranziehen um eine Einschätzung zu treffen, wie die Auslastung des Systems aussieht. Die Zahlen

bei `load average` geben an, wieviele CPU-Kerne in den letzten 1, 5 und 15 Minuten ausgelastet waren, die Zahl alleine ist ohne zu wissen, wieviele Kerne im System vorhanden sind nicht sehr aussagekräftig, so wäre ein System mit einer 4-Kern CPU schon mehrfach überlastet und Prozesse würden auf CPU-Ressourcen warten, ein System mit 72 CPU-Kernen hätte noch freie Kapazitäten. In der Zeile mit `%Cpu(s)` sind verschiedene Lastparameter aufgeführt die nachfolgend erklärt sind.

```
us  Last für laufende Nutzerprozesse
sy  Last für Betriebssystemprozesse
ni  Last für niedrigpriorisierte Prozesse (NI/nice > 0)
id  freie CPU-Zeit
wa  Wartezeit für freie I/O (Eingabe/Ausgabeprozesse)
hi  Last für Hardwareinterrupts
si  Last für Softwareinterrupts
st  Last für virtualisierte (Gast)betriebssysteme
```

Der `nice`-Wert gibt die Priorität für den Systemprozessscheduler an und kann von `-20` bis `19` annehem, wobei 0 der Standardwert ist. Je höher der Wert ist, desto niedriger ist die Priorität des Prozesses und umgekehrt. Erhöhte Prioritäten kann nur ein Nutzer mit `root`-Rechten zuweisen. Informationen über die im System verbaute(n) CPU(s) erhält man mit dem Befehl `lscpu` oder `/proc/cpuinfo`, Informationen über den Arbeitsspeicher erhält man mittels `free -m`, `/proc/meminfo`. Mit `numactl --hardware` kann man sich einen groben Überblick über CPU-Kerne und Arbeitsspeicher und die Zuordnung zueinander verschaffen. Im schon angeführten `/proc`-Verzeichnis ist für jeden laufenden Prozess ein Ordner mit der `${PID}$` abgelegt, über den man Informationen zu diesem laufenden Prozess bekommen kann.
Wie schon zuvor erwähnt, ist in Unix alles eine Datei, um zu sehen, welche Dateien derzeit offen sind kann der Befehl `lsof` verwendet werden, damit kann man zum Beispiel auch herausfinden, ob zum Beispiel ein Port schon von einer Anwendung geöffnet ist, und eine weitere Anwendung diesen Port nicht mehr verwenden kann. Hier als Beispiel eines `sshd`-Wurzelprozess.

```
[root@centos ~]# lsof -p 1463
COMMAND  PID USER   FD   TYPE   DEVICE SIZE/OFF    NODE NAME
sshd    1463 root  cwd    DIR    253,0     4096       2 /
sshd    1463 root  rtd    DIR    253,0     4096       2 /
sshd    1463 root  txt    REG    253,0   877760 4480401 /usr/sbin/sshd
(...)
sshd    1463 root  mem    REG    253,0   123336 4475561 /usr/lib64/libaudit.so.1.0.0
sshd    1463 root  mem    REG    253,0   278504 4458889 /usr/lib64/ld-2.28.so
sshd    1463 root    0r   CHR      1,3      0t0    1027 /dev/null
sshd    1463 root    5u  IPv4    36489      0t0     TCP *:ssh (LISTEN)
sshd    1463 root    7u  IPv6    36491      0t0     TCP *:ssh (LISTEN)
```

Man sieht hier, dass sowohl für die Protokolle `IPv4` und `IPv6` der Port `ssh` geöffnet ist - ein Blick nach `/etc/services` gibt einem die Portnummer 22 - und allen lokalen Netzwerkadressen zugeordnet ist - erkennbar am Stern `*` vor dem `:22` - und dass die Datei /usr/sbin/sshd,

/dev/null sowie eine ganze Anzahl an weiteren Bibliotheken verwendet wird.
Möchte man einen Prozess beenden, so kann man dafür `kill ${PID}` oder `pkill ${Muster}` verwenden, wenn man das Recht dafür hat. Hierbei kann man verschiedene Signale angeben, mit denen ein Prozess beendet werden soll. Alle verfügbaren Signale kann man in `man 7 signal` nachlesen.
Programme, die noch weiterlaufen sollen, nachdem man sich von dem System abgemeldet hat startet man mittels `nohup`, Hintergrundprozesse kann man mittels `&` starten, diese beiden Befehle kann man auch kombinieren. Prozesse, die man aktiv laufend in den Hintergrund schicken möchte kann man mit `[STRG]+[z]` pausieren und mit `bg` in den Hintergrund schicken. Im hintergrund laufende Jobs kann man mit `jobs` abfragen und mit `fg ${job-ID}` wieder in den Vordergrund holen.
Die tiefere Performanceüberwachung ist ein ganz eigenes Thema und soll hier nicht weiter behandelt werden.

11 Shellskripting

Arbeiten auf der Kommandozeile können auch gescripted werden, das heißt ich fasse mehrere Befehle in einer Datei zusammen, die ich dann ausführe und die dann diese Befehle ausführt. In dieser Datei können wie in einer Programmiersprache Schleifen und Fallentscheidungen verwendet werden und die Rückmeldung der Programme können auch in Arrays weiterverwendet werden. Dies gibt einem ein mächtiges Werkzeug in die Hand.
Eine Shellscript-Datei hat in der ersten Zeile einen sogenannten Shebang `#!` gefolgt von der Shell, mit der das gesamte Skript ausgeführt werden soll. Das Shellskript muss das Ausführungsflag `x` für den Benutzer haben - sonst muss man das Shellskript als Parameter mit dem Binary einer Shell aufrufen, zum Beispiel `/bin/bash test.sh`. Außerdem ist angeraten die Dateinamenerweiterung `.sh` anzugeben, damit man mit einem `ls` sieht, dass es sich um ein Shellskript handelt.

Nachfolgend ein Skript, dass sowohl Schleifen, als auch Arrays verwendet und auch Berechnungen anstellt, um die aktuelle CPU-Kernfrequenz auszugeben. Die Ausgabe kann dann zum Beispiel von Überwachungstools wie `nagios` oder `checkmk` ausgewertet werden.

```
[slommage@centos ~]# cat ./get_cpuspeed.sh
#!/bin/bash

for i in `ls -1 /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq`
do
    let "myspeed=((`cat $i`/1000))"
    cpuspeed+=( $myspeed )
done

for ((i = 0; i < ${#cpuspeed[*]}; i++))
do
    printf ${cpuspeed[$i]}" "
done
printf "\n"
[slommage@centos ~]# ./get_cpuspeed.sh
2858 2816 2599 2800 3003 2899 2873 2916
```

Der Befehl `let` führt eine Berechnung durch, in dem Fall eine Division durch 1000. Mit dem `$`-Zeichen wird auf den Inhalt einer Variable referenziert, das `#cpuspeed[*]` ermittelt die Anzahl der Elemente im array `cpuspeed`.
Der Übergang von Shellskripting zur Notwendigkeit der Verwendung von richtigen Programmiersprachen ist fließend und muss vom Nutzer selbst eingeschätzt werden.

Literatur

[1] D. M. Ritchie; K. Thompson, *The UNIX Time-Sharing System* (April 3, 1978), 1905–1929.

[2] `https://web.archive.org/web/20151220013149/http://www.novell.com/news/press/archive/1995/12/pr95274.html`. Accessed December 20, 2015.

[3] `https://web.archive.org/web/20160919232940/http://www.zdnet.com/article/twenty-years-of-linux-according-to-linus-torvalds/`. Accessed September 19, 2016.

[4] `https://w3techs.com/technologies/overview/operating_system`. Accessed September 28, 2021.

[5] `https://top500.org/statistics/sublist/`. Accessed September 28, 2021.

[6] `https://web.archive.org/web/20171011232456/https://www.greenbot.com/article/3138394/android/report-nearly-90-percent-of-smartphones-worldwide-run-android.html`. Accessed October 11, 2017.

[7] `https://gs.statcounter.com/os-market-share/desktop/worldwide/`. Accessed September 28, 2021.

[8] `https://image.slidesharecdn.com/vicheatsheetv100-111203142712-phpapp01/95/vi-cheat-sheet-v-1-00-1-1024.jpg?cb=1322922811`. Accessed September 29, 2021.

[9] `https://www.bell-labs.com/usr/dmr/www/pdfs/man71.pdf`. Accessed October 3, 2021.

[10] https://www.regex101.com. Accessed October 3, 2021.

[11] https://www.etsi.org/deliver/etsi_ts/119300_119399/119312/01.02.01_60/ts_119312v010201p.pdf. Accessed October 3, 2021.

[12] https://datatracker.ietf.org/doc/html/rfc4251. Accessed October 5, 2021.