How to UNIX

Erste Schritte in der Welt von UNIX

Malte Grave

11.09.2022

Inhaltsverzeichnis

1 Einführung

1.1 Was ist UNIX?

Dieses Skript wird dich in die wundervolle Welt von UNIX einführen bzw. dich auf den Vortrag vorbereiten. Aber zunächst müssen wir uns einmal die Frage stellen, was überhaupt UNIX ist und warum Ihr das hier überhaupt lesen solltet. Um es kurz und einfach zu fassen, ist UNIX nichts anders als ein Betriebssystem. Ihr kennt bereits ein Betriebssystem, die Chance ist hoch, das Ihr dieses Skript gerade auf Windows lest. Hinter UNIX steckt eine lange Geschichte, die ist aber irrelevant für euch. Wichtig ist für euch nur, das es heute viele unixartige Betriebssysteme gibt.
Ihr kennt doch sicher diese "Hacker" aus Filmen oder? Grüner Text auf dem Bildschirm und nur Zahlen- bzw. Buchstabensalat, der klassische Hacker eben, selbstverständlich darf die Kapuze nicht fehlen!
Doch was Ihr dort seht ist tatsächlich mehr oder weniger real, zumindest ein Teil davon. Oft werden in diesen Filmen unixartige Betriebssysteme benutzt.
Ihr habt doch bestimmt mal von Linux gehört? Das ist dieses Betriebssystem mit dem drolligen Pinguin, es ist auch ein unixartiges System.
Aber die Frage bleibt, wofür braucht Ihr den Quatsch jetzt?! Das ist ganz einfach zu beantworten. Es gibt Dinge, die auf z.B Windows gar nicht funktionieren und oder nur schwer zu erreichen sind. Große Servercluster verwenden kein Windows, weil dieses darauf nie ausgelegt worden ist. Klar gibt es Windows für Server, aber wenn es z.B um wissenschaftliches Rechnen geht, werdet Ihr keine Windows Server mehr sehen. Das gleiche gilt für Unmengen von Severapplikationen, seien es Webserver, Gameserver oder auch Mailserver. Auf der ARBI (*Abteilung Rechner- und Netzbetrieb Informatik*) läuft das unixartige Betriebssystem **FreeBSD** und das schauen wir uns jetzt genauer an, aber zuerst müssen wir noch klären, wie wir uns mit den Servern der ARBI verbinden.

1.2 Das Terminal

Auf Windows kennt ihr Fenster. Ihr könnte diese mit der Maus öffnen, verkleinern, vergrößern oder auch schließen. Aber wie sieht das auf unixartigen Systemen aus? Von Vorne weg, es gibt auch dort Fenster. Sogennante *window manager* stellen diese Funktionen bereit. Aber das ist doch langweilig, wir wollen was besseres benutzen, womit wir die volle Kontrolle haben und genau wissen was gerade passiert. Dafür benutzen wir das Terminal oder zu deutsch auch Konsole. Wenn Ihr euch mit der ARBI verbindet *(dazu mehr im Kapitel 2)* könnt Ihr quasi ausschließlich, mit dem Terminal auf dem Server arbeiten. Eine grafische Oberfläche bleibt euch weitestgehend erspart. Das Terminal ermöglicht uns eine sehr nahe Kommunikation mit dem Betriebssystem, uns steht damit die Welt von UNIX offen.

2 Die Secure Shell

2.1 Das Protokoll SSH

Um mit einem Server zu kommunizieren, benötigt es ein Protokoll welches bestimmte Aufgaben erledigt. Wir wollen mit dem Server über Text kommunizieren, wir haben keine bunten tollen Benutzeroberflächen. Alles läuft bei uns über die Tastatur ab. Aber selbstverständlich soll die Verbindung auch noch sicher sein. Daten sollen nicht über Klartext versendet werden.

Und genau das ermöglicht uns **SSH**, sogar noch mehr! Das sehen wir aber erst später. Die Sicherheit ist ein großes Thema bei SSH, die Authentifizierung verläuft über das Public-Key Verfahren (erlernt Ihr im Modul **Diskrete Strukturen)**.

SSH basiert dazu auf einer Server-Client Architektur. Unser PC/Laptop wäre also der Client und die ARBI der Server.

Uns stehen mehrere Server von der ARBI zur Verfügung. Die wichtigsten für Studierende nutzbare Server sind hier aufgeführt.

Domain	IP
duemmer.informatik.uni-oldenburg.de	134.106.11.89
weser.informatik.uni-oldenburg.de	134.106.11.83
ems.informatik.uni-oldenburg.de	134.106.11.84
kuestenkanal.informatik.uni-oldenburg.de	134.106.11.40

2.2 Der SSH Client

2.2.1 Windows

Unter Windows bietet sich Putty sehr an. Es ist ein sehr schlankes Programm, welches uns mit (fast) allen Features, die SSH unterstützt versorgt. Einschränkungen gelten beim sogenannten *X11 forwarding*, damit ist es möglich grafische Anwendungen über das SSH Protokoll darstellen zu lassen. Unter Windows funktioniert dies aber nur mit Umwegen. Ich gehe hier darauf nicht näher ein.

Beginnen wir erst einmal mit der Installation. Hier gibt es zwei Wege. Entweder könnt Ihr Putty "hart"auf dem System installieren oder nur als einzelne Binary herunterladen, die Ihr irgendwo auf dem System platzieren könnt. Zur offiziellen Seite von Putty gelangt Ihr hier.

Ich empfehle euch die Binary herunterzuladen, Ihr findet diese unter dem Reiter **"Alternative binary files"**. Wählt einfach die 64-bit Variante. Danach könnt Ihr Putty ganz normal Starten. Ihr werdet dann folgendes Fenster erhalten:

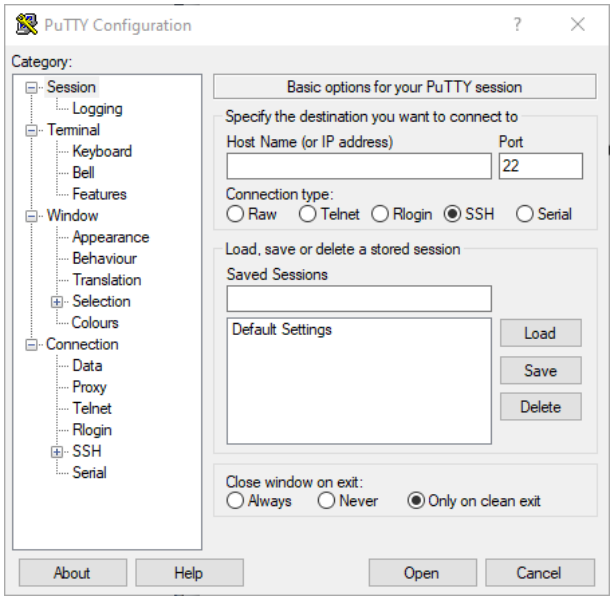

Abbildung 1: Der Putty SSH Client für Windows

In der Zeile "*Host Name (or IP adress)*" könnt Ihr aus der obigen Tabelle, entweder die Domain oder die IP eintragen, je nach dem auf welchen Server Ihr euch einloggen wollt. Anschließend klickt Ihr unten auf "*Open*".

Folgend sollte eine Meldung auftauchen die so in etwa aussieht.

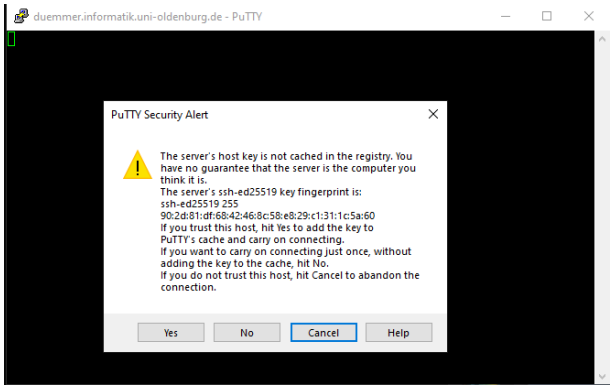

Abbildung 2: Key caching für extra Sicherheit

Hier könnt Ihr beruhigt auf "**Yes**" klicken. Dies dient dazu, das wir den Server **eindeutig** in unsere *.know_hosts* Datei mitaufnehmen. So garantieren wir, dass es sich um den gleichen Server handeln muss, mit welchem wir uns verbinden möchten bzw. auch zukünftig verbinden wollen. Als nächstes findet die Authentifizierung mit dem Server statt. Ihr werdet nach eurem Benutzernamen und Passwort gefragt (Die Daten erhaltet

Ihr von uns während des Vorkurses). **Wichtig ist, das auf allen Server die gleichen Benutzerdaten gültig sind!**

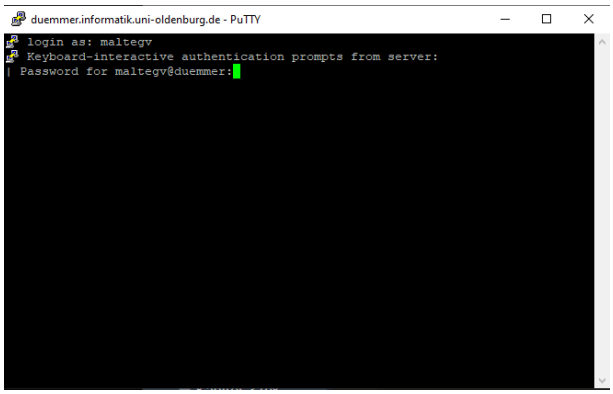

Abbildung 3: Passwort abfrage bei Putty

Wenn Ihr erfolgreich authentifiziert worden seid, könnt Ihr nun Text in die Konsole tippen. Das, was Ihr nun seht, wird als Shell bezeichnet. Wir sind jetzt in der Lage, mit dem Betriebssystem (Kernel) zu kommunizieren. Das ganze sieht dann so aus:

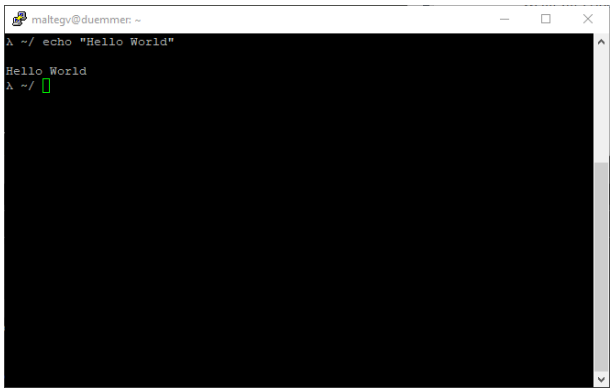

Abbildung 4: Beispiel Eingabe bei Putty

Probiert es einfach selber mal aus, gibt folgendes in die Shell ein:

Das $ signalisiert einen Shell-Befehl! Bitte gibt es nicht mit in den Befehl ein. Das ist eine Art Konvention, damit jeder weiß, dass es sich um einen Shell-Befehl handelt.

```
$ echo "Hello World"
```

Als Ausgabe sollte dann wie oben im Bild folgendes stehen:

```
Hello World
```

Herzlichen Glückwunsch, das war euer erster Shell-Befehl!

2.2.2 GNU/Linux

Auf jeder (gänigen) Linux Distributionen ist ein Terminal vorinstalliert. Und meistens auch der SSH Client. Öffnet dazu einfach euer Terminal und tippt

```
$ ssh -V
```

ein. Ihr solltet die Version eures SSH Client (*meistens OpenSSH*) im Terminal angezeigt bekommen. Falls Ihr eine Meldung wie "*Command not found*" bekommen solltet, installiert den SSH Client einfach nach. Je nach Linux Distribution unterscheiden sich die Installationen. Hier werde ich nicht näher drauf eingehen.

Die Verbindungsherstellung verläuft auch recht einfach. Gibt einfach folgendes dazu im Terminal ein.

```
$ ssh <username>@<server-ip or domain>
```

2.2.3 MacOS

Falls Ihr auf einem Mac unterwegs seid, habt Ihr es auch recht einfach, wie die GNU/-Linux Fraktion. Auf jedem Mac ist ein Terminal vorinstalliert, sowie der OpenSSH SSH-Client. Ich empfehle jedoch die Installation von iTerm2, da dieses viele Vorteile im Vergleich zum "Standard Terminal" bietet.

Was die Verbindung zum Server mithilfe des SSH-Clients betrifft, macht Ihr genau das gleiche wie eure GNU/Linux Kollegen.

```
$ ssh <username>@<server-ip or domain>
```

2.2.4 Abschlusswort

Ihr wisst jetzt wie Ihr euch mit einem SSH-Server verbindet, jetzt steht der Vortrag über UNIX an. Ich empfehle euch Notizen währenddessen zu machen und anschließend während den Übungen Fragen zu stellen, wenn es unklarheiten gibt. Es kann ganz schön überwältigend für den Anfang sein, so viel unbekanntes Zeug zu sehen. Deswegen keine Scheu! Wir haben alle mal klein angefangen. ;)

3 Unix Tutorial

3.1 Eine Einfürung

Wir werden erst einmal grundlegend starten. Auf der Agenda stehen, das Filesystem von UNIX, die Navigation, die Erstellung und die Manipulation von Dateien. Danach werden wir uns die ARBI einmal genauer anschauen, welche möglichkeiten wir haben und viel wichtiger, wie Ihr euch dort zurechtfindet. Am Schluss gibt es noch eine kleine Einführung in das sehr mächtige Shell-Scripting. Los geht's!

Loggt euch nun wie oben beschrieben mit euren Nutzerdaten an einen der ARBI-Server per SSH ein, danach könnt Ihr direkt loslegen.

3.1.1 Das Filesystem

Wenn Ihr euch erfolgreich eingeloggt habt, könnt Ihr nun Text in das Terminal eingeben (wie z.B beim obigen *Hello World* Beispiel). Vielleicht erscheint es auf Anhieb erst einmal suspekt, aber Ihr befindet euch jetzt in einem Ordner.
Und zwar in einem ganz speziellem, eurem **Persönlichem Ordner**! In der UNIX-Welt sagen wir dazu auch **home directory** oder auch kurz **home**. Hier werdet Ihr euch wahrscheinlich am häufigsten befinden, da Ihr hier alle Berechtigungen habt und euch damit selbstverständlich austoben könnt. Und genau das bringt uns zu einem der wichtigsten Punkte in UNIX. Das ***Fileystem***.

Als erstes schauen wir uns an wie wir Ordner erstellen um unsere Daten schön zu organisieren. Später werde ich die generelle Struktur von UNIX veranschauen.

Wir erstellen jetzt einen Ordner mit dem Namen *Dokumente*, dafür tippt Ihr folgendes in das Terminal ein.

```
$ mkdir Dokumente
```

Das war es schon. Doch was habt Ihr jetzt genau gemacht? Ihr habt im Terminal den Befehl *mkdir* aufgerufen. Konkret heißt *mkdir* **make directory**. Klingt ja erst mal ganz logisch. Anschließend habt Ihr dem Befehl aber noch etwas weiter mitangegeben, und zwar den Namen des zu erstellenden Ordners. Das nennt sich *Argument*, in unserem Fall war es der Name *Dokumente*.
Befehle nehmen *Argumente* und *Optionen* entgegen, dazu aber später mehr.

Doch wie sehen wir nun ob wir den Ordner tatsächling erstellt haben? Wir haben ja schließlich kein Feedback bekommen. Zum glück kann UNIX uns auch dort mit einem Befehl uns weiterhelfen. Um alle Dateien in einem Order anzuzeigen gibt es den Befehl

```
$ ls
```

ls steht für **list**.

Aber moment mal! Wieso habe ich gerade geschrieben, das wir uns alle Dateien in einem Ordner mit *ls* anschauen können? Wir haben doch schließlich gerade einen Ordner erstellt und keine Datei oder?

In der UNIX-Welt gibt es einen ganz wichtigen und entscheiden Satz, dieser Lautet:

„**Everything is a file**“.

Ihr müsst es nicht auf Anhieb verstehen wieso, weshalb und warum. Es reicht wenn Ihr es euch Merkt, dass alles in UNIX Dateien sind. Das erklärt auch warum wir mit *ls* unseren Ordner anzeigen können.

Aber wie genau kommen wir jetzt in unseren neu erstellen Ordner? Wir sind ja immer noch in unserem **home** Verzeichnis. Dreimal dürft Ihr Raten. Genau, dafür gibt es auch einen Befehl.

```
$cd <dirname>
```

Ersetzt *<dirname>* durch *Dokumente*, danach befindet Ihr euch in eurem erstellen Ordner. Um zurück zum eurem **home** Ordner zu kommen, folgendes eingeben.

```
$ cd ..
```

.. bedeutet ein Ordner weiter *hoch* zu Wandern. Stellt es euch wie eine Treppe vor. Ihr befindet euch auf der Mitte der Treppe und geht eine Stufe runter. Danach wollt ihr wieder nach oben indem Ihr wieder eine Stufe aufsteigt.

Kommen wir jetzt zu den bereits erwähnten *Optionen* von Befehlen. Ihr kennt bereits den Befehl *ls*. Es gibt auf UNIX versteckte Dateien. Diese folgen einer ganz einfachen Regel. Beginnt eure Datei mit einem . , werden sie versteckt.

Probiert es mal aus, erstellt einen Ordner mit dem Namen **.Dokumente** und gibt danach den Befehl *ls* ein. Was seht Ihr? Richtig nur euren "alten" Dokumente Ordner.

Wie können wir uns aber nun unseren verstecken Ordner anzeigen lassen? Wir nutzen dafür die bereits erwähnten *Optionen*. Um versteckte Dateien mit *ls* anzeigen zu lassen fügt ihr am Ende von *ls* ein *-l* an. Das sieht dann folgend so aus.

```
$ ls -l
```

Nun seht Ihr viele verschiedene Dateien, die euch davor nie angezeigt worden waren. Keine Sorge, die könnt ihr erst einmal ignorieren. Wir werden uns als nächstes mit

dem genrellen Aufbau (Ordnern) von UNIX befassen, bevor wir weiter mit Dateien herumspielen.

Aus Windows kennt Ihr die berühmte **C:** Festplatte, auch **root** genannt. Dort befindet sich alles, eure Windows Installation, Programme, Dokumente, Spiele und vieles mehr. In Unix ist das sehr ähnlich, auch wenn wir hier nicht mit **C:** arbeiten. Was **C:** in Windows ist, ist in UNIX ganz einfach nur / .
Das Prinzip ist aber das gleiche. Unter / finden sich auch Ordner wieder (dazu gleich mehr), wie bei Windows.

Angenommen wir befinden uns jetzt auf einem frisch Installierten UNIX-System und euer Benutzer heißt **test**, dann wäre eurer **home** unter **/home/test** zu finden.
Unter Windows wäre es analog folgender Pfad: **C:\Users\test**.

Bleiben wir aber mal bei der Annahme, das wir uns auf einem nicht modifiziertem UNIX Server befinden, der gerade aufgesetzt wurde. Den wir schauen uns nähmlich jetzt die Ordner Struktur in / an.

Das sind im wesentlichen die wichtigsten Order, die (eigentlich) immer vorhanden sind. Unter dem Ordnerbaum erkläre ich jeden Ordner einmal, warum und wofür bzw. welchen nutzen dieser hat.

```
/
├── bin
├── etc
├── home
├── usr
├── lib
├── var
└── tmp
```

bin dort finden wir die Standard Befehle wie *mkdir*, *ls*, *cp* und *mv*. **bin** steh für *binary*.

etc dort finden wir Konfigurationsdateien und Informationsdateien.
etc steht für **editable text configuration**

home dort finden sich alle Benutzer mit ihrem persönlichem Ordner wieder. Name ist selbsterklärend.

usr hier finden sich z.B alle weiteren, dazu installierten Programme und auch Biblotheken. **usr** steht für *user* (war damals so, **home** gab es früher noch nicht).

lib wichtige Systembibliotheken sind hier vorhanden, aber auch Bibliotheken die von anderen Programmen verwenden werden. **lib** steht für **library**.

var enthält Datein die gerne geändert werden u.A durch Programme. Ein Beispiel wären Logs. **var** steht für **variables**.

tmp in tmp sind Datein erhalten die nicht langlebig sind und schnell wieder gelöscht werden. **tmp** steht für **temporary**.

Weitere Informationen zu der Ordnerstruktur sind im Ubuntuusers Wiki zu finden.

3.1.2 Dateien kopieren und manipulieren

Nun schauen wir uns an, wie wir z.B eine Textdatei erstellen können und diese anschließend Manipulieren (umbenennen, bearbeiten oder auch löschen) können. Es gibt einen Textbasierten Editor Namens *nano* auf UNIX, dieser ermöglicht uns das Editieren von Text. Wir erstellen uns jetzt ein Datei mit dem Namen **Meine-Datei.txt**.

Dafür geben wir jetzt folgendes in die Konsole ein.

```
$ nano Meine-Datei.txt
```

Anschließend befindet Ihr euch im Editor selbst. Ihr könnt jetzt irgendwas dort hineinschreiben. Wenn Ihr dann fertig seid und die Datei speichern wollt, drück die Tasten $STRG + x$, danach bestätigt Ihr mit der Taste y das Ihr die Datei speichern wollt.

$\gamma_1 + \delta_v \models x^2$

3.1.3 Die ARBI erkunden

3.2 Shell-Scripting