Einführung in UNIX

Vortrag für das Proseminaer UNIX-Tools von Andreas Amereller

A.Was ist UNIX?

UNIX ist ein Mehrbenutzer- und Mehrprozessbetriebssystem. UNIX wird meistens als Akronym „UNIX is not MultiCS" interpretiert. Die erste Version (1969, damals noch UniCS) entstammt den Bell Laboratories und war als Alternative zu dem Betriebssystem MultiCS gedacht. 1977 folgte dann das erste BSD (Berkeley Software Distribution) der Universität in Berkeley, Kalifornien. Danach folgten 1981 das System III und 1983 das System V von den Bell Labs. Heute existiert eine Vielzahl von UNIX-Derivaten, z.B. Linux, Sun Solaris, SGI IRIX, Free-, Open-, Net-BSD, BeOS, MacOS X etc., die wiederum für verschiedene Prozessorarchitekturen verfügbar sind, z.B. x86, AMD64, PowerPC, Sparc, Arm, IA64, Alpha etc. Letztendlich basieren alle UNIX-Derivate auf der BSD- oder System-V-Linie.
Inzwischen erfreut sich UNIX großer Beliebtheit und gilt als wichtigstes Betriebssystem für Workstations.
UNIX-Betriebssysteme gibt es sowohl als kommerzielle Versionen, als auch als „OpenSource", d.h. Distributionen, bei denen der gesamte Quellcode mitgeliefert wird und die in den meisten Fällen auch kostenlos sind.

B.Grundlagen von UNIX-Systemen

1.Das Mehrbenutzersystem

Um ein UNIX-System benutzen zu können muss man zuerst eine Sitzung anmelden und sich als dazu berechtigter Benutzer ausweisen. Dieser Anmeldevorgang wird im „Computerdeutschen" auch als einloggen bezeichnet.
Jede Benutzerkennung ist folgendermaßen aufgebaut:

- Benutzername
- Passwort
- Gruppenzugehörigkeiten
- Login-Shell

Um sich also anmelden zu können, braucht man eine Benutzerkennung (User Account). Beim Anmeldevorgang wird nun zuerst der Benutzername angegeben, gefolgt von einem Passwort.
Die Gruppenzugehörigkeiten dienen der Zugriffsregelung für Benutzer und Gruppen. Die beruht auf dem Rechtesystem von UNIX, das für jede Datei des Systems verschiedene Rechte für Benutzer, Gruppe und Sonstige vergibt (wird weiter unten erläutert). Auf diese Weise ist es gerade in Firmen möglich, dass mehrere Benutzer auf die Dateien ihrer Gruppe(n) Zugriff haben.

Nach erfolgreicher Anmeldung wird die Login-Shell gestartet und der Benutzer kann das System benutzen.

2. Dateistruktur und Rechtesystem

Anders als bei MSDOS oder MS Windows ist die Dateistruktur von UNIX streng hierarchisch. Eigentlich kann man sagen, dass das ganze System ausschließlich aus Dateien besteht. Auch die Verzeichnisse und selbst alle Geräte wie Soundkarte, Festplatten, Maus, Tastatur etc. Sind Dateien, auf die das System zugreift. Ebenfalls wie alles Andere im System sind auch alle Prozesse und auch der gesamte Arbeitsspeicher als Dateien im Verzeichnis */proc* gespeichert. Die komplette Dateistruktur beginnt beim Wurzelverzeichnis (/). Desweiteren werden die Festplatten etc. Nicht über Laufwerksbuchstaben o. ä. Definiert. Stattdessen werden innerhalb des Dateibaums eingehängt (gemountet). So ist beispielsweise das CD-ROM-Laufwerk auf /mnt/cdrom gemountet. Neben dieser Struktur existiert in jedem UNIX-System ein Rechtesystem, das für jede Datei gilt. Dieses Rechtesystem ist folgendermaßen aufgebaut:
Jede Datei hat drei grundlegende Rechte:

- Lesen
- Schreiben
- Ausführen

Jedes dieser Rechte kann für drei Bereiche gesetzt werden:

- Benutzer
- Gruppe
- Sonstige

Mithilfe dieses Rechtesystems kann jeder Benutzer genau festlegen, wer welchen Zugruff auf welche Datei haben soll.
Dies hilft auch der Privatsphäre bei Software. Jeder Benutzer kann Software nach Belieben für sich installieren oder deinstallieren, ohne dass diese andere Benutzer oder gar das System beeinträchtigt. Ausserdem kann natürlich auch hier der Benutzer private Software so einrichten, dass nur er darauf zugreifen kann.

3. Prozessstruktur und -hierarchie

Nach den Dateien sollten auch noch die Prozesse Erwähnung finden. Als Prozess wird ein gerade im Arbeitsspeicher ablaufendes Programm definiert, das Systemresourcen wie Arbeitsspeicher und Prozessorleistung belegt. Auch hier liegt wieder eine strenge Hierarchie vor. Zuerst wird in jedem UNIX-System der Init-Prozess gestartet, der Vaterprozess für alle weiteren Prozesse ist. Dieser Prozess startet nach und nach, direkt oder indirekt, alle Prozesse, die das System braucht. So ist auch die Login-Shell, die bei der Anmeldung gestartet wird, ein Nachfolger des Init-Prozesses. Diese „Vater-Kind”-Beziehung besteht

grundsätzlich und ohne Ausnahme zwischen allen Prozessen. Wenn also z.B. ein Benutzer sich anmeldet und aus seiner Login-Shell (Bsp. Bash) das Programm „programm-eins" startet, so sieht die Hierarchie so aus:
Init->login->bash->programm-eins

4. Der Systemverwalter

Der Systemverwalter, auch root-Benutzer oder einfach nur „root", ist der einzige Benutzer auf einer UNIX-Plattform, der ***alles*** darf. Er hat uneingeschränkten Zugriff auf alle Dateien aller Benutzer, kann als einziger deren Zugehörigkeiten und/oder Rechte ändern und kann Benutzer und Gruppen erstellen, verändern oder löschen. Desweiteren kann ausschließlich „root" Systemsoftware installieren oder deinstallieren. Dies gewährleistet, dass jeder Benutzer die Software, die er braucht oder haben will, sicher installieren kann, ohne dass sie ohne die Zustimmung vom Systemverwalter auf das System zugreifen kann. Natürlich obliegt dem Systemverwalter auch die wichtigste Aufgabe des Systems: das System in Stand zu halten. Ausserdem genießt der root-Benutzer das vollständige Vertrauen aller Benutzer, da er ja sämtliche Dateien einsehen und gegebenenfalls verändern kann. Da es nur einen root-Benutzer gibt, teilen sich heute oft mehrere Leute die root-Kennung. Dies erleichtert zum Einen die Instandhaltung und Wartung des Systems, andererseits kann es einen Missbrauch der root-Kennung einschränken. Es kann aber auch ein Sicherheitsrisiko bedeuten, da man nicht weiß, wer als „root" was tut.

5. Die wichtigsten UNIX-Befehle

Um ein UNIX-System gut bedienen zu können, sollte man einige der Grundbefehle jedes UNIX-Systems kennen, wenngleich heutzutage eine Vielzahl von grafischen Benutzeroberflächen existiert, die gerade Anfängern den Umgang mit UNIX-Systemen erleichtern. Trotzdem ist eine Shell fast immer unentbehrlich. Im Folgenden sollen diese Befehle und Programme aufgelistet und kurz erklärt werden. Diese Liste umfasst Befehle zum anzeigen von Dateien und deren Inhalte, die Navigation im Dateibaum und einige Befehle für „root":

- **logout**
  Der logout-Befehl ist deshalb wichtig, da es immer empfohlen wird, seine Sitzung zu beenden, auch wenn man den Arbeitsplatz nur kurz verlässt. Dies soll verhindern, dass in der Abwesenheit ein Anderer private Daten einsehen oder verändern oder die Benutzerkennung auf irgend eine andere Weise missbrauchen kann.

- **ls**
  „ls" ist einer der wichtigsten Befehle. Er listet alle Dateien und Verzeichnisse im angebenen oder aktuellen Verzeichnis auf, wenn

kein Verzeichnis angegeben wurde.
Beispiel: ls /home/andy/bsp

```
ausfbsp1 beispiel1 ordner1
```

Hier wird also der Inhalt des Verzeichnisses */home/andy/bsp* aufgelistet.
Die häufigsten Parameter, die bei „ls" vorkommen, sind:

- **-l**
  „-l" bewirkt eine genaue Angabe der aufgelisteten Einträge. So werden bei jedem Eintrag sämtliche Rechte, der Eigentümer, dessen Gruppe, die Größe des Eintrags, das Erstellungsdatum und der Name angegebn.
  Beispiel: ls -l /home/andy/bsp

```
-rwxr-xr-- andy users 24681 20.8.2005 ausfbsp1
-rw-r----- andy users 12345 20.8.2005 beispiel1
drwxr-x--- andy users 13579 25.8.2005 ordner1
```

  In diesem Beispiel ist der erste Eintreg eine ganz normale Datei namens *beispiel1*, die dem Benutzer *andy* und der Gruppe *users gehört. Sie ist 12345 bytes groß und wurde am 20.8.2005 erstellt. Der Benutzer (andy) hat Lese- und Schreibrchte, die Gruppe (users) hat Leserechte, alle anderen haben keinerlei Zugriffsrechte. Die zweite Datei ist ausführbar für den Benutzer und die Gruppe, alle anderen haben ausschließlich Leserechte. Der dritte Eintrag ist ein Verzeichnis, das der Benutzer und die Gruppe öffnen dürfen,alle anderen haben keine Zugriffsrechte.*
- **-a**
  Dieser Parameter listet auch versteckte Dateien, also Deteien, die mit einem Punkt („**.**") beginnen, auf.
- **--color=auto**
  Bei vielen UNIX-Systemen hat „ls" die Fähigkeit, die verschiedenen Dateitypen zu erkennen und farblich zu markieren, was die Lesbarkeit erheblich erhöht. Dieser Parameter aktiviert die Farbmarkierung.
- **-h**
  „-h" gibt die Dateigröße mit einem „Größenbuchstaben" nach der Zahl aus, sodass die Größe besser lesbar ist. Die Buchstaben sind z.B. K für kilobyte, M für megabyte etc.

- **find**
  Der find-Befehl sucht im angegebenen Verzeichnis nach Dateien. Optional können noch viele weitere Parameter angegeben werden. Um den Rahmen nicht zu sprengen, wollen wir uns hier auf die Verzeichnisangabe und die Parameter -type und -iname beschränken:
  - ***Verzeichnis***
    Hier wird das Verzeichnis angegeben. Wird hier nichts angegeben, sucht „find" im aktuellen Verzeichnis.
  - **-type *typ***
    Es werden nur Dateien vom Typ *typ* aufgelistet. Die wichtigsten Typen sind:

- **f** Datei ist eine reguläre Datei
- **d** Datei ist ein Verzeichnis
- **l** Datei ist ein symbolischer Link

- **-(i)name „*name*”**
  Nur Dateien, in denen die Zeichenfolge *name* vorkommt, werden aufgelistet. Bei *-name „name”* wird auf Groß- und Kleinschreibung geachtet, bei *-iname „name”* wird sie nicht beachtet

Bsp.: *find /home/andy/bsp -type d -iname „OrdNEr1”*
Diese Eingabe sucht im Verzeichnis „/home/andy/bsp” alle Verzeichnisse namens „OrdNEr1” ohne Berücksichtigung der Groß- und Kleinschreibung. Die Ausgabe sieht dann so aus:
/home/andy/bsp/ordner1

- **pwd**
  Dieses Kommando gibt das aktuelle Verzeichnis aus. Auf diese Weise fällt es leicht, sich zurechtzufinden, sollte man vergessen haben, in welchem Verzeichnis man sich gerade befindet.

- **chdir**
  Um in der Shell gut navidieren zu können, wird man um diesen Befehl nicht herumkommen, da dieser Befehl für Verzeichniswechsel zuständig ist. In vielen UNIX-Varianten ist er auch als „cd” implementiert. Die Syntax ist:
  cd *Verzeichnis*

  Für *Verzeichnis* ist „.”, „..” oder jeder andere Verzeichnispfad zulässig. So bewirkt beispielsweise *cd ..* einen Wechsel in das nächsthöhere Verzeichnis, *cd /home/andy/bsp* wechselt ins Verzeichnis „/home/andy/bsp” und *cd .* bewirkt nichts (bzw. wechselt vom aktuellen ins aktuelle Verzeichnis).

- **mkdir**
  mkdir existiert in einigen UNIX-Varianten auch als „md” und legt ein neues leeres Verzeichnis an. Hierbei kann der Name ein absoluter oder relativer Pfad sein.
  Bsp.:
  - ***mkdir ordner2* (relativer Pfad)**
    Hier wird im aktuellen Verzeichnis ein leeres Verzeichnis namens „ordner2” erstellt.
  - ***mkdir /home/andy/bsp/ordner1/unterordner1* (absoluter Pfad)**
    Hier wird im Verzeichnis „/home/andy/bsp/ordner1” ein leeres Verzeichnis namens „unterordner1” erstellt.

- **cp**
  Mit „cp” werden Dateien von einer Quelle an ein Ziel kopiert. Auch hier kann sowohl für *Quelle* als auch für *Ziel* ein relativer oder absoluter Pfad angegeben werden. Es können sowohl Dateien als auch Verzeichnisse angegeben werden. Sollte als Quelle oder Ziel das Verzeichnis „**.**” angegeben werden, so wird vom/ins aktuelle Verzeichnis kopiert. Die wichtigsten Parameter sind „-f” und „-r”.
  - **-f**

Dieser Parameter forciert den Kopiervorgang und ignoriert sämtliche Warnungen. Dieser Parameter sollte nur angewendet werden, wenn man sich vollkommen sicher ist.

- **-r**
  Diese Option führt den Vorgang rekursiv aus, also auch auf alle Unterverzeichnisse und deren Dateien. Dies ist sinnvoll, wenn man z.B. mehrere Verzeichnisse kopieren will.

- **mv**
  Der mv-Befehl funktioniert ähnlich wie „cp", nur, dass er Dateien nicht kopiert sondern verschiebt. Die Dateien werden also an der alten Stelle gelöscht bzw sie werden umbenannt. Wie bei „cp" stehen auch hier unter Anderem die Optionen „-f" zum Forcieren und „-r" zum rekursiven Verschieben zur Verfügung. Auch die Syntax ist gleich wie bei „cp":
  mv [-rf] *Quelle Ziel*

- **rm**
  Der rm-Befehl löscht Dateien und Verzeichnisse. Die Syntax ist gleich wie bei „cp" und „mv":
  rm [-rf] *Ziel*
  Hier wird also die Datei/das Verzeichnis „Ziel" gelöscht.
  !Achtung!
  Sollte man „root" sein, muss dieser Befehl **mit äußerster Vorsicht** eingesetzt werden. Sollte man nämlich „rm -rf /" eingeben wird ***das gesamte System vernichtet***, da ALLE Dateien im Verzeichnisbaum gelöscht werden. Sollte man gar den rm-Befehl mit den Optionen **-Rrf** ausführen, so ist das System **in jedem Fall** verloren, da „-R" auch die Oberverzeichnisse mitlöscht.

- **less**
  Der Befehl „less" ist deshalb sehr nützlich, da man mit ihm Dateien ansehen kann. Allerdings ist „less" kein Editor, d.h. Man kann Dateien ansehen, sie aber nicht verändern. Dafür gibt es Programme wie beispielsweise *Vi/Vim* (wird später behandelt). Die Syntax ist:
  less *Dateiname*

- **chmod**
  Dieser Befehl verändert die Zugriffsrechte für Dateien und Verzeichnisse. „chmod" kann nur auf Dateien angewendet werden, die dem ausführenden Benutzer gehören. Ein Benutzer kann also keine Dateien eines anderen oder gar des „root"-Benutzers umstellen. Die Rechte werden folgendermaßen editiert:
  chmod *BRGRSR Ziel*

  - **BR**
    Mit BR bezeichne ich hier die Rechte des Eigentümers. Diese Option ist folgendermaßen aufgeschlüsselt:
    4 -- Leserecht
    2 -- Schreibrecht
    1 -- Ausführungsrecht
    Soll der Benutzer also eine Datei lesen, schreiben und ausführen

können muss man hier „7” eingeben. Will man dagegen nur lesen und ausführen, gibt man „5” ein.

- **GR**
  GR bezeichnet hier die Gruppenrechte. Auch hier kann aus den drei Optionen beliebig kombiniert werden.
- **SR**
  SR steht für die Rechte aller anderen, die nicht zur Gruppe gehören. Wie bei BR uns SR kann hier aus den oben genannten Optionen beliebig kombiniert werden.

Gibt man an einer der drei Stellen „0” ein, so hat diese Stelle gar keine Zugriffsrechte auf diese Datei.
Beispiel: chmod 640 meinedatei
Diese Eingabe ändert die Datei „meinedatei” folgendermaßen:
Der Eigentümer kann die Datei lesen und schreiben, seine Gruppe kann sie nur lesen und alle anderen können sie weder lesen noch schreiben. Ausführen kann diese Datei niemand.
Sollte man die Rechte eines Verzeichnisses ändern, sollte man bedenken, dass man ein Verzeichnis nur öffnen kann, wenn es als ausführbar gekennzeichnet ist.
Auch für „chmod” existiert eine Option „-R”, mit der man „chmod” rekursiv ausführt.
Ausnahme: root
Der root-Benutzer genießt auch hier wieder eine Sonderrolle. Da er der Systemadministrator ist, kann auf alle Dateien beliebig zugreifen und ihre Zugriffsrechte ändern.

- **chown**
  Dieser Befehl ist ausschließlich „root” vorbehalten. „chown” ändert den Eigentümer und die Gruppe einer Datei oder eines Verzeichnisses. Mit diesem Befehl kann „root” z.B. einem Dateimissbrauch entgegenwirken oder Dateien, z.B. Teile von Software in der Entwicklung auf andere Entwickler „umschreiben”

- **man**
  Dieser Befehl ruft, soweit vorhanden, eine „Manpage” zu dem angegebenen Programm auf. In diesen „Manpages” stehen alle wichtigen Informationen zu dem gesuchten Befehl oder Programm.
  Die Syntax ist folgende:
  man *prog*
  Hier ruft „man” die manpage des Programms *prog* auf. Sollte keine manpage verfügbar sein, wird dies auf dem Bildschirm ausgegeben.

- **apropos**
  Der Befehl „apropos” sucht in allen manpages nach dem Wort bzw. Der Zeichenfolge, die als Parameter an „apropos” angehängt wird.
  Beispiel: apropos *„change directory”*
  Auf diesen Befehl hin werden alle manpages am Bildschirm aufgelistet, in denen der Ausdruck „change directory” vorkommt.

C. Ausblicke

Natürlich gibt es unter UNIX noch eine ganze Reihe weiterer Befehle wie „grep”, „ps”, „kill”, „top” etc. Diese sind allerdings erst für versiertere Benutzer von Bedeutung oder im UNIX-Alltag von geringer Wichtigkeit..
Desweiteren ist man in einem UNIX-System nicht nur auf die Konsole angewiesen. Inzwischen gibt es eine Vielzahl von grafischen Benutzeroberflächen, wie z.B. CDE, Gnome, KDE, Windowmaker, IceWM, fvwm etc. Diese GUIs (**G**raphical **U**ser **I**nterface) bringen ihrerseits wieder Programme mit, die entweder die oben aufgeführten Befehle benutzen oder diese ersetzen.
Da UNIX permanent weiterentwickelt wird und ursprünglich als Entwickler-System gedacht war, stehen dem Benutzer auch einige weitere Tools zur Verfügung. Dazu gehören Compiler und Interpreter für verschiedene Programmiersprachen wie C/C++, Python, Perl, Ruby, Java etc., die „Sprache” (La-)Tex und viele weitere Softwarepakete wie Maple, OpenOffice, StarOffice, vi, emacs.

Literatur-Angabe:

www.linuxhq.com/guides/LUG/node14.html (Geschichte von UNIX, Prozesse)

Das restliche UNIX-Wissen entstammt dem täglichen UNIX/Linux-Gebrauch und der intensiven Benutzung der man-pages.