Universität Potsdam
Institut für Informatik
Wintersemester 2011–2012

Grundlagen von Betriebssystemen

Aufgabenblatt 4

Der Editor `vi` – Übung

1. Kopieren Sie die Datei `/home/rlehre/da1` in Ihr Arebitsverzeichnis und öffnen Sie diese Datei mit dem Editor `vi`. Editieren Sie die Datei, wobei Sie den Anweisungen in der Datei `da1` folgen. Speichern Sie das Ergebnis als Datei `da2` in Ihrem Arbeitsdverzeichnis, *ohne* den Editor zu beenden.

2. Probieren Sie aus und notieren Sie sich die Antworten.
   Was bewirkt die Eingabe von `/`*Zeichenkette*`<Enter>` im Kommandomodus des `vi`?

   Was bewirkt `//` ? _____

   Welches dieser Kommandos funktioniert in `more`, welches in `less`?

   ---

6 Die Shell: Der UNIX-Kommandointerpreter

Die folgenden Aufgaben sind davon abhängig, dass Sie eine Tenex-C-Shell (tcsh) verwenden. Einige Aufgaben beziehen sich auf Besonderheiten in der Umgebung im Sun-Pool (`babylon`-Domäne). Wenn Sie zu Hause arbeiten, sollten Sie eine SSH-Verbindung z.B. zu `delenn.babylon.cs.uni-potsdam.de` herstellen und dort die Aufgaben bearbeiten.

3. *Der Kommandointerpreter, den Sie benutzen, ist die sogenannte Tenex-C-Shell, die mit jedem Öffnen eines Terminal- oder Konsolenfensters automatisch gestartet wird. Es gibt eine ganze Reihe weiterer Programme zur Kommandointerpretation (Shells), die Sie per Kommando starten können, z.B. die C-Shell (Kommando* `csh`*), Bourne-Shell (*`sh`*), Bourne-Again-Shell (*`bash`*) usw.*

   (a) Starten Sie diese Shells nacheinander *als Subshells (von der ursprünglichen Shell abgeleitete Shells)*, indem Sie die entsprechenden Kommandos eingeben. Starten Sie dann noch einmal eine Tenex-C-Shell (Kommando: `tcsh`), und sehen Sie sich nun Ihre Prozesse mit `ps -f` an. Achten Sie auf die PIDs und die PPIDs!

   (b) Beenden Sie nun die Shells wieder mit `exit`. Zählen Sie mit: Sie benötigen also viermal `exit`, um zu Ihrer ersten Tenex-C-Shell zurückzugelangen.

   (c) Was passiert, wenn Sie ein fünftes `exit` eingeben? _____

   Hinweis: Die verschiedenen Shells unterscheiden sich durch ihren Kommandovorrat und die Wirkungsweise einzelner Kommandos und Shell-Mechanismen. Wir konzentrieren uns auf die Tenex-C-Shell.

4. *Jede Shell läuft in einer definierten* Umgebung*, die für die Funktionsweise der Shell und der von ihr aufgerufenen Programme wichtig ist. Die Umgebung wird mit Hilfe von Variablen abgespeichert. Man unterscheidet* **globale und lokale Variablen**.

   (a) Fragen Sie die lokalen Variablen mit `set` und dann die globalen Variablen mit `env` ab. Welchen Wert haben folgende globale Variablen?

   `HOST` _____

   `USER` _____

   `GROUP` _____

   (b) Finden Sie eine globale Variable, die nicht auch als lokale Variabele existiert und umgekehrt (von Groß-/Kleinschreibung abgesehen).

   _____

   (c) Führen Sie aus: `echo $HOME`. Was wird also bei `echo $`*variablenname* ausgegeben.

   _____

   (d) Legen Sie eine benutzerdefinierte lokale Variable an: `set v = wert1`
   Legen Sie eine benutzerdefinierte globale Variable an: `setenv U wert2`
   Überzeugen Sie sich mit `set` und `env`, dass die Variablen korrekt angelegt wurden. Greifen Sie auch mit dem `$`-Mechanismus auf `v` und `U` zu.

   (e) Öffnen Sie jetzt eine neue Tenex-C-Shell als Subshell. Welche Variablen werden von der Elternshell an die Subshell weitergegeben?

   lokale Variablen (ja/nein): _____ globale Variablen (ja/nein): _____
   (*Hinweis: Die Systemvariablen, die automatisch angelegt werden, können Sie zur Beantwortung dieser Frage nicht heranziehen. Achten Sie auf benutzerdefinierte Variablen.*)

   (f) Löschen Sie `U` mit `unsetenv U`. Kann man noch auf `U` zugreifen? _____

   (g) Beenden Sie die Subshell. Existiert `U` noch in der Elternshell? _____

   (h) Löschen Sie `v` mit `unset v`. Kontrollieren Sie!

   (i) *Info: Die Unterscheidung von globalen und lokalen Variablen ist deshalb wichtig, weil die meisten Programme, die eine Shell aufruft, in einer Subshell ausgeführt werden. Dies gilt auch für Programme, die von bereits laufenden Programmen aufgerufen werden.*
   Legen Sie eine eigene lokale und eine eigene globale Variable an. Starten Sie jetzt den Editor `vi` und führen Sie im `vi` UNIX-Kommandos mit `:!`*Kommando* aus. Greifen Sie dabei mit `echo $`*Variablenname* einmal auf Ihre globale Variable und einmal auf Ihre lokale Variable zu. Was stellen Sie fest? _____

   _____

   Wie erklären Sie sich das? _____

5. Beschäftigen Sie sich mit der Rolle einiger Systemvariablen!

   (a) Laden Sie die Datei `Systemvariablen.pdf` herunter und lesen Sie sie durch!

   (b) Um den absoluten Pfad der Systemprogramme herauszufinden, kann man `which` benutzen. Notieren Sie die Pfade der Kommandos `cd, firefox, mount`:

   `which ls` _____ `which firefox` _____ `which mount` _____

   (c) Probieren Sie diese Kommandos noch einmal, aber verwenden Sie jetzt `where` anstelle von `which`. Können Sie alle von `where` angezeigten Programme starten, wenn Sie nicht nur den Kommandonamen, sondern den absoluten Pfad am Eingabeprompt eingeben?
   Probieren Sie außerdem `/usr/sbin/mount`. Dies sollte eine Ausgabe erzeugen.

Info: Das Kommando `mount` *dient dem Systemadministrator dazu, verschiedene Dateisysteme zu einem virtuellen Dateisystem zusammenzubauen (s. Einführungsvorlesung). Ohne Argumente wird gewissermaßen der "Bauplan" Ihres virtuellen Verzeichnissystems angezeigt, also welcher Verzeichnisname mit welchem physikalischen (Teil-)Dateisystem verknüpft ist. Sehen Sie sich die Ausgabe ruhig einmal genauer an und achten Sie z.B. auf die tatsächliche Lokalität Ihrer Login-Verzeichnisse.*

(d) Erschließen Sie die Bedeutung von `which` und `where`, indem Sie den Wert der `PATH`-Variablen ansehen.

i. Warum erzeugt `where mount` keine Ausgabe? _____

ii. Warum zeigt `which ls` gerade diese Datei an, und nicht die andere, die bei `where ls` ausgegeben wird? _____

(e) Ergänzen Sie den Wert von `PATH` wie folgt: `setenv PATH ${PATH}:/usr/sbin`
Überprüfen Sie, ob sich der Wert von `PATH` wie gewünscht verändert hat.
Erzeugen `which` und `where` jetzt Ausgaben mit dem Argument `mount`?

(f) Erzeugen Sie ein Unterverzeichnis `MyHome` und ändern Sie den Wert von `home` wie folgt:
`set home = ~/MyHome`
Überprüfen Sie Ihre Änderung und führen Sie dann `cd` (ohne Argument) aus. In welchem Verzeichnis sind Sie gelandet (`pwd`)? Warum?

_____

(g) Haben sich die "gespiegelten" Variablen (`path` und `HOME`) mit verändert? _____

(h) Schließen Sie Ihr Terminalfenster. Öffnen Sie ein neues. Wie sind jetzt die Werte von `HOME` und `PATH`? Warum? _____

_____

6. Ändern Sie Ihren Umgebungsbereich dauerhaft!

(a) *Die Konfiguration Ihrer Systemumgebung erfolgt beim Login durch Abarbeitung verschiedener Dateien mit Kommandostapeln (Skripte), die sich u.a. im Verzeichnis* `/etc` *(einheitliche Einstellungen für alle Benutzer) und in Ihrem Login-Verzeichnis (individuelle Einstellungen) befinden.*
Betrachten Sie die Dateiinhalte der Skripte `.login` und `.tcshrc`, die als versteckte Dateien in Ihrem Login-Verzeichnis existieren. In welchem der Skripte wird hauptsächlich die Systemumgebung (globale Variablen), in welchem die Shell (lokale Variablen) konfiguriert? Umgebung: _____ Shell: _____

(b) *Wenn Sie Modifikationen an der Systemkonfiguration vornehmen wollen, sollten Sie (um ein "Reset" zu ermöglichen), alle Änderungen in seperaten Dateien organisieren und in* `.login` *bzw.* `.tcshrc` *nur für den Aufruf dieser neuen Dateien sorgen.*

i. Wechseln Sie in Ihr Login-Verzeichnis!
Erzeugen Sie (mit einem beliebigen Texteditor, z.B. mit dem `vi`) eine neue Datei `.login.private`, die als einzige Zeile einen `setenv`-Befehl enthält. Es soll die `PATH`-Variable so ergänzen, dass das aktuelle Arbeitsverzeichnis immer zuletzt nach ausführbaren Dateien durchsucht wird.

Kommando: _____
(Was glauben Sie, wozu das nützlich sein könnte?)

ii. Editieren Sie jetzt die Datei `.login` so, dass eine neue letzte Zeile mit dem Inhalt
`source .login.private`
erscheint.

iii. Führen Sie `.login` aus, indem Sie auf der Kommandozeile `source .login` eingeben. Überprüfen Sie den veränderten Wert der `PATH`-Variablen.

(c) *Es könnte auch gewünscht sein, dass Kommandos wie* `cp` *oder* `rm` *immer mit Sicherheitsabfrage funktionieren, also wie mit Option* `-i`*, aber ohne diese Option immer angeben zu müssen. Dazu dient der Alias-Meachnismus der Shell.*

i. Das Kommando `alias` (ohne Argumente) zeigt Ihnen alle Alias-Vereinbarungen an, die für Sie gesetzt sind. Es sollte bei Ihnen eine leere Ausgabe erzeugen. Sonst wiederholen Sie die Aufgabe 1 aus dem 3. Aufgabenblatt.

ii. Erzeugen Sie zwei leere Dateien. Führen Sie aus: `alias rm 'rm -i'`.
Führen Sie jetzt noch einmal `alias` aus, löschen Sie danach eine Ihrer leeren Dateien mit `rm` (ohne Option). Sie erhalten die Sicherheitsabfrage.

iii. Heben Sie für *einen* Befehlsaufruf den Alias-Mechanismus auf: Löschen Sie die zweite leere Datei mit `\rm` (mit vorangestelltem Backslash!). Überzeugen Sie sich, dass der vereinabrte Alias trotzdem noch aktiv ist.

iv. Heben Sie die Alias-Vereinbarung dauerhaft auf: `unalias rm`. Überprüfen Sie!

v. Editieren Sie die (zur Zeit leere) Datei `.aliases` in Ihrem Loginverzeichnis, so dass in einem Kommandostapel die Namen `cp, rm, mv` dauerhaft so eingestellt sind, dass Sie ohne Angabe von `-i` mit Sicherheitsabfrage funktionieren. (*Hinweis: Für die automatische Ausführung der Datei* `.aliases` *wird bereits gesorgt.*) Öffnen Sie ein neues Terminalfenster und überzeugen Sie sich, dass Ihre Vereinbarungen in `.aliases` gesetzt wurden. Fragen Sie bei Problemen eine Lehrkraft!

7. Arbeiten Sie mit Shell-Substitutionen!
Sie haben bereits gesehen, dass das `$`*-Zeichen eine Sonderbedeutung in der Shell hat: Bevor ein Kommando ausgeführt wird, substituiert die Shell zunächst alle Zeichenkette, die mit* `$` *beginnen, durch die Werte derjenigen Variablen, deren Namen hinter dem* `$`*-Zeichen beginnt. Es gibt weitere Shell-Substitutionen.*

(a) Wechseln Sie in `/bin` und führen Sie aus:

`ls *sh`; Ausgabe: _____

`ls ?sh`; Ausgabe: _____

`ls [az]sh`; Ausgabe: _____

`ls [a-m]sh`; Ausgabe: _____

Interpretieren Sie die Ergebnisse mit dieser Übersicht zur **Dateinamenexpansion**:

Wildcard	Bedeutung
`*`	beliebeige Zeichenkette (auch die leere)
`?`	ein beliebeiges Zeichen
`[...]`	ein Zeichen aus der Menge, z.B. `[aeiou]`
`[`*von*`-`*bis*`]`	ein Zeichen zwischen *von* und *bis*
`[^...]`	ein Zeichen außerhalb der Menge, z.B. `[^0-9]`
`^...`	Negation der gesamten Wildcard
{*wort1,...,wortn*}	eines der Wörter
`~`	absoluter Pfadname des (eigenen) Login-Verzeichnisses
`~`*user*	abs. Pfadname des Login-Verzeichnisses von *user*

(b) Wechseln Sie in Ihr Login-Verzeichnis. Werden bei `ls *` alle Dateien angezeigt? Probieren Sie auch `ls .*`. Wie können Sie sich alle Dateien in Ihrem Login-Verzeichnis mit nur einem `ls`-Kommando ausgeben lassen?

_____

(c) Wechseln Sie in das Verzeichnis `/usr/bin`. Geben Sie nun mit Hilfe des `echo`-Kommandos alle Dateienamen in `/usr/bin` aus, die

i. die Länge 2 haben,

ii. mit einem Vokal beginnen und auf `d` oder `s` enden,

iii. mindestens eine Ziffer enthalten.

Das `echo`-Kommando soll jeweils nur ein Argument haben!

(d) Führen Sie folgende Kommandozeilen aus, die eine **Kommandosubstitution** enthalten und notieren Sie die jeweilige Ausgabe:

i. `echo Ich arbeite auf dem System` `` `uname` `` . _____

ii. `echo Die Systemzeit ist` `` `date` `` . _____

(e) Welchen Wert hat die Variable `vlist` nach `set vlist =` `` `ls` `` ? _____

_____

(f) Welche ASCII-Zeichen haben Sie insgesamt kennengelernt, die eine Sonderbedeutung in der Shell besitzen?

_____

Um diese Zeichen vor einer Interpretation durch die Shell zu schützen, können Sie sie auf drei Arten maskieren:

`\`	schützt das direkt folgende Zeichen,
`'...'`	schützt alle eingeschlossenen Zeichen,
`"..."`	schützt alle eingeschlossenen Zeichen außer `\, $, !` und `` ` ``

(g) Geben Sie mit `echo` aus: `Die Zeichen * und $home werden interpretiert.`

Kommando mit 1. Maskierungsart: _____

Kommando mit 2. Maskierungsart: _____

Ausgabe bei der 3. Maskierungsart: _____

(h) Geben Sie im Verzeichnis `/home/rlehre` alle Dateien aus, deren Namen ein Fragezeichen enthält.

Kommando: _____