Programmierkurs IPK SS23
Andreas Naumann
IWR, Universität Heidelberg

Übung 1
Abgabedatum —
ipk-exercises:2023-ss

Übung 1

Aufgabe 1: Kommandozeile

(a) Benutzen Sie die Kommandozeile, um unter Verwendung der in der Vorlesung vorgestellten Befehle `cd` und `mkdir` in Ihrem Home-Verzeichnis ein Verzeichnis `uebungen` und darin ein Verzeichnis `uebung01` anzulegen.

```
~ $ mkdir uebungen
~ $ cd uebungen
uebungen $ mkdir uebung01
uebungen $ cd uebung01
uebung01 $
```

Wenn Sie nähere Informationen über einen Befehl möchten, schauen Sie sich die Dokumentation des Befehls, die sogenannte *manpage* mit dem Befehl `man BEFEHL` an, z.B. `man mkdir`. Sie können in der Ausgabe mit den Pfeiltasten scrollen und mit der Taste `"q"` zur Kommandozeile zurückkehren.

(b) Erstellen Sie in diesem Verzeichnis mit einem Texteditor Ihrer Wahl die Datei `helloworld.cc` mit folgendem Inhalt:

```cpp
#include <iostream>

int main(int argc, char** argv)
{
  std::cout << "Hello world!" << std::endl;
  return 0;
}
```

Kompilieren Sie das Programm und führen Sie es aus. Zum Kompilieren verwenden wir hier den Gnu compiler `g++`, die Option `"-o <name>"` sagt dem Compiler, wie das erzeugte Programm heißen soll.

```
uebung01 $ g++ -o helloworld helloworld.cc
uebung01 $ ./helloworld
Hello world!
uebung01 $
```

Hinweis: Es ist sinnvoll Warnmeldungen beim Compilieren zu aktivieren. Ersetzen sie dafür `"g++"` durch `"g++ -Wall"`. Für den Fall, dass Sie den `clang` compiler verwenden können sie entsprechend `"clang++ -Wall"` benutzen.

(c) Probieren Sie aus, was passiert, wenn Sie die Option `"-o <name>"` weglassen. Zur Erinnerung: Sie können den Inhalt des aktuellen Verzeichnisses mit dem Befehl `ls` anzeigen. Versuchen Sie, das vom erzeugte Programm ohne ein vorangestelltes `"./"` auszuführen.

(d) Geben Sie die Datei `helloworld.cc` mit Hilfe des Programms `cat` auf der Kommandozeile aus. Was passiert, wenn Sie das gleiche mit dem kompilierten Program `helloworld` versuchen?

(e) Suchen Sie mit Hilfe von `grep` in der Datei `/usr/include/stdio.h` nach dem Begriff *FILE*. Probieren Sie auch die Option `"-n"` aus.

grep wort datei.cc

(f) Suchen Sie mit Hilfe von `grep` in der Datei `helloworld.cc` nach *Hello world*. Was beobachten Sie?

(g) Die Shell trennt Argumente, sobald ein Leerzeichen auftaucht. Sie können dies umgehen, indem Sie Argumente, die aus mehreren Wörtern bestehen, in Anführungszeichen setzen. Versuchen Sie, den Befehl aus dem vorhergehenden Aufgabenteil so zum Laufen zu bringen.

Aufgabe 2: Shell 2

Nachdem Sie sich auf dem letzten Übungsblatt mit grundlegenden Funktionen der Shell vertraut gemacht haben, geht es heute um neue Befehle und etwas fortgeschrittenere Features.

Hinweis: Denken Sie daran, dass Sie bei neuen Befehlen die manpage lesen können oder Hilfe zum Befehl im Internet suchen können. Das Finden und Verstehen derartiger Informationen ist eine wichtige Fähigkeit, die Sie trainieren sollten!

(a) Benutzen Sie die Kommandozeile, um die Datei https://conan.iwr.uni-heidelberg.de/data/teaching/ipk_ws2018/exampleprogram.tar.xz herunterzuladen. Hierfür können Sie den Befehl

```
curl -LO <url>
```

verwenden, wobei Sie `<url>` durch die herunterzuladende Adresse ersetzen.

(b) Die heruntergeladene Datei ist ein komprimiertes *TAR*-Archiv, ähnlich einer ZIP-Datei unter Windows. Manchmal haben diese Dateien auch `.gz` oder `.bz2` als Endung, dies hängt davon ab, wie die Datei komprimiert wurde.

Überprüfen Sie, ob die Datei wirklich ein *TAR*-Archiv ist. Hierfür können Sie den Befehl

```
file <dateiname>
```

verwenden, der die Datei untersucht und das Ergebnis auf der Konsole ausgibt.

(c) Entpacken Sie die Datei mit dem Befehl

```
tar xf <dateiname>
```

Schauen Sie sich an, welche Verzeichnisse und Dateien durch das Entpacken angelegt wurden.

(d) Kompilieren Sie die im Archiv enthaltene C++-Datei `exampleprogram.cc` und wählen Sie die Compiler-Optionen so, dass das entstehende Programm `exampleprogram` heisst. Führen Sie das Programm aus und finden Sie heraus, was es macht.

(e) Ersetzen Sie `exampleprogram.tar.xz` durch ein neues Archiv, das zusätzlich zu `exampleprogram.cc` auch das ausführbare Programm enthält. Hierfür wechseln Sie zurück in das Verzeichnis, das das heruntergeladene Archiv enthält und verwenden den Befehl

```
tar cfJ <Archivname> <Datei/Verzeichnis>...
```

Dieser Befehl legt ein neues Archiv mit dem angegebenen Namen an und speichert darin alle aufgelisteten Dateien und Verzeichnisse. Letztere werden dabei mit allen enthaltenen Dateien und Unterverzeichnissen hinzugefügt.

(f) Schauen Sie sich den Inhalt des Archivs mit

```
tar tf <Archivname>
```

an. Danach verschieben Sie die Archivdatei in ein neues Verzeichnis (legen Sie ein neues an, falls nötig) und entpacken Sie das verschobene Archiv wieder. Versuchen Sie, das enthaltene Programm `exampleprogram` auszuführen.

Aufgabe 3: Kommandozeile für Fortgeschrittene

Diese Aufgaben sind für die Fortgeschritteneren unter Ihnen, die sich schon mit der Shell auskennen. Um diese Aufgaben zu lösen, müssen Sie eventuell die *man pages* der Befehle lesen und / oder im Internet nach Tips suchen.

(a) Finden Sie mit Hilfe der Shell, Pipes und den Befehlen `find` und `wc` heraus, wie viele C-Header-Dateien (Endung `.h`) sich im Verzeichnis `/usr/include` und allen Unterverzeichnissen befinden.

(b) Finden Sie mit Hilfe der obigen Befehle sowie `xargs` und `grep` mit einer passenden *regular expression* heraus, wie viele *include statements* sich in diesen Dateien befinden. Include statements bestehen aus einer Zeile, die mit `#include` beginnt, allerdings können vor dem `#include` noch beliebig viele Leerzeichen stehen.

Aufgabe 4: Versionskontrolle mit git

git ist, wie in der Vorlesung vorgestellt, ein verteiltes Versionskontrollsystem. In dieser Übung geht es darum, sich ein wenig mit dem System vertraut zu machen.

(a) Melden Sie sich bei einem der Dienste GitLab[1], Bitbucket[2] oder GitHub[3] an.

(b) Erstellen Sie auf der Website ein neues Projekt für die Übungen, z.B. `ipk-exercises`, und folgen Sie den Anweisungen auf der Website, um dieses Repository auf Ihren Computer zu klonen.
Wichtig: Auf GitLab folgen Sie den Schritten nur bis zum Ende des Absatzes “Create a new repository”!

(c) Kopieren Sie die Quellcodedateien der bisherigen Übungen in das neue Verzeichnis, fügen Sie sie zu git hinzu und erstellen einen Commit. Beim Erstellen des Commits müssen Sie eine Commit Message eingeben. Dies können Sie auf zwei verschiedene Arten machen:

- Sie schreiben einfach `git commit`. In diesem Fall öffnet sich im Terminal ein Texteditor mit einigen auskommentierten Zeilen (beginnend mit `"#"`). Schreiben Sie in die erste (leere) Zeile die Commit Message (diese kann auch mehrere Zeilen lang sein), speichern Sie die Datei und verlassen Sie den Editor. Meistens öffnet sich unter Linux der Editor `vim`, der etwas gewöhnungsbedürftig ist:
  1. Drücken Sie die Taste `"i"` (für “insert”), um den Eingabemodus zu aktivieren.
  2. Tippen Sie die commit message.
  3. Drücken Sie die Taste `"ESC"`, um den Editiermodus zu verlassen.
  4. Geben Sie folgendes ein, um die Datei zu speichern und den Editor zu schliessen: `":wq"` und drücken Sie Enter.
- Alternativ können Sie die commit message auch auf der Kommandozeile eingeben:

```
1 git commit -m "hier steht Ihre commit message"
```

  Damit ist es allerdings schwierig, mehrzeilige Commit Messages zu schreiben.

Hinweis: Kompilierte Programme sollten normalerweise nicht eingecheckt werden, da man diese ja einfach durch erneutes Kompilieren wieder aus den Quellcodedateien erzeugen kann.

(d) Pushen Sie den Commit auf den Server.

(e) Während der Arbeit in den Übungen entstehen diverse Dateien, die nicht mit unter Versionskontrolle sollen, z.B. Sicherungsdateien, `.o`-Dateien und CMake-Buildverzeichnisse. Um diese Dateien zu *ignorieren*, gibt es in git den `gitignore`-Mechanismus (siehe `git help gitignore`).

[1] https://gitlab.org
[2] https://bitbucket.org
[3] https://github.com

Erstellen Sie in Ihrem Arbeitsverzeichnis eine Datei `.gitignore` (Achtung, diese Datei ist versteckt, zum Anzeigen verwenden Sie `ls -a`) mit Dateinamen, die nicht zu git hinzugefügt werden sollen, z.B.

```
# alle Dateien, die auf .o enden
*.o
# eine bestimmte Datei, z.B. ein Executable
statistics
# Sicherungskopien, die auf ~ enden
*~
# das Unterverzeichnis build/ (von CMake)
build/
```

Hierbei werden Zeilen, die mit `#` beginnen, wieder als Kommentare ignoriert. Fügen Sie die Datei zu git hinzu, erstellen Sie einen Commit und pushen Sie diesen.

Von nun an können Sie git verwenden, um Dateien zwischen Ihrem Laptop und dem Rechner im Computer-Pool zu synchronisieren.

(f) Testen Sie, ob alle wichtigen Dateien auf dem Server angekommen sind (am einfachsten über das Webinterface). **Danach** löschen Sie das lokale Verzeichnis mit Ihren Übungen, indem Sie in das übergeordnete Verzeichnis wechseln und den Befehl `rm -rf ipk-exercises` ausführen, wobei Sie `ipk-exercises` durch den Namen ersetzen müssen, den Sie für das Verzeichnis verwendet haben.

Im Anschluss klonen Sie das Repository wieder vom Server und überprüfen, dass alle eingecheckten Dateien weiterhin da sind.

Hinweis: Auch in Zukunft ist es sinnvoll, Ihre Dateien zumindest am Ende der Übungsstunde zu committen und auf den Server zu pushen. Dann können Sie das Repository von zu Hause erneut klonen und an den Übungen weiterarbeiten. Es ist auch eine gute Idee, regelmäßiger Commits zu machen, z.B. immer dann, wenn man eine Teilaufgabe erfolgreich gelöst hat.

Weitere Hilfe zu git finden Sie unter anderem hier:

- https://git-scm.com/doc Ausführliche Dokumentation und einige Einführungsvideos.
- https://git-scm.com/docs/everyday Eine praktische Kurzübersicht wichtiger Befehle.
- https://git-scm.com/docs/gittutorial Das offizielle Tutorial.

Ansonsten ist Google Ihr Freund...