**Versionsmanagement:**

**Erstellen einer diff-Datei:**

- jeder Hunk (außer bei Dateianfang) beginnt mit einer nicht veränderten Zeile und besitzt ansonsten nur veränderte, gelöschte oder hinzugefügte Zeilen

- "!" steht für eine geänderte Zeile, "+" für eine hinzugefügte und "-" für eine entfernte Zeile

- ein Hunk hat die Form:

\*\*\* Anfang in source, Ende in source \*\*\* Hunk Hunknummer \*\*\* unveränderte Zeile in source und dest veränderte oder gelöschte Zeile in Source (veränderte oder gelöschte Zeile in Source) ... \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --- Anfang in dest, Ende in dest --- unveränderte Zeile in source und dest veränderte oder hinzugefügte Zeile in dest (veränderte oder hinzugefügte Zeile in dest) ... \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Dreiwegeverschmelzung:**

Annahme: Basisversion (B) und Nachfolgeversionen (NV1 und NV2) sind bekannt und Ergebnisversion (E) soll ermittelt werden.

Die Grundidee ist, dass wenn in genau einer der beiden Nachfolgeversionen eine Zeile verändert wurde oder wenn in beiden Nachfolgeversionen die gleiche Veränderung gemacht wurde, dann wird diese Veränderung übernommen. Wenn bei einer Zeile keine Veränderung vorgenommen wurde, dann wird die Zeile aus der Basisversion übernommen. Wenn eine Zeile in NV1 und NV2 unterschiedlich verändert wird oder an der gleichen Stelle 2 unterschiedliche Textzeilen eingefügt werden, dann muss manuell ein Konflikt behoben werden.

Regeln: Eine Textzeile ist in den 3 Versionen gleich ==> wird in E übernommen Textzeile ist in B, aber nicht in NV1 und/oder NV2 ==> wird nicht in E übernommen Textzeile ist in NV1 und/oder NV2, aber nicht in B ==> wird in E übernommen Textzeile in NV1 und NV2 verändert oder unterschiedliche Zeilen an der gleichen Stelle eingefügt ==> manuelle Konfliktbehebung

**Buildmanagement:**

Aufbau Abhängigkeitsbeschreibung: ziel: objekt1, objekt2, objekt3, ... Kommando(s) zur Erzeugung von ziel aus Objekten

Beispiel (wandelt img.jpg in img.png um): img.png : img.jpg convert img.jpg img.png

Makros: Wird am Anfang des make-files zum Beispiel "*CON=convert*" definiert, so wird bei Ausführung des make-files jedes Vorkommen von ${CON} bzw. $(CON) durch convert ersetzt.

Sonderzeichen-Makros: beispiel.png: beispiel.jpg convert beispiel.jpg beispiel.png

$@ Ziel der Regel mit Suffix (z.B. beispiel.png) $\* Ziel der Regel ohne Suffix (z.B. beispiel) $^ alle Objekte einer Regel rechts von ":" (z.B. beispiel.jpg) $? alle Objekte einer Regel, die neuer als Ziel sind $< erstes Objekt rechts von ":"

$(wildcard Muster) liefert alle Dateien mit bestimmten Muster zurück $(wildcard \*.png) liefert alle Dateien mit der Dateiendung .png zurück

$(patsubst Suchmuster, Ersatzstring, Liste von Wörtern) $(patsubst %.png, %.jpg, $(wildcard \*.png)) liefert für jede .png Datei einen String, in welchem das .png durch .jpg ersetzt wurde zurück

% steht für den gleich bleibenden Teil der Strings

hinter "default : " steht der Befehl, der beim ausführen des make-files ausgeführt wird

Aufgaben: - Alle expandierten Systembefehle auflisten ==> von "default" aus alle Makros durch Strings ersetzen. Die Systembefehle sind die Kommandos (siehe Aufbau Abhängigkeitsbeschreibung), die nicht als Ziel in einer der Abhängigkeiten vorkommt. - Was passiert, wenn make-file erneut aufgerufen wird (ohne Änderung)? Warum? ==> Nichts, da beim Auflösen der Abhängigkeiten der Zeitstempel überprüft wird und damit erkannt wird, dass die Dateien bereits auf dem neuesten Stand sind.

Tipps: - wenn in der Lücken ausgefüllt werden sollen und eine Lücke vor eine öffnenden Klammer ist, dann gehört mit sehr hoher Wahrscheinlichkeit ein $ davor - bei Systembefehlen wie cc steht hinter den Flags das Ziel vor der Liste von verwendeten Objekten

**Variantenmanagement:**

? bedeutet optional ! bedeutet benötigt X bedeutet m aus n <=> bedeutet 1 aus n <=> Requires: Element bedeutet es wird Element benötigt Durchfahrt verboten Schild bedeutet exclude Abhägigkeit

(muss noch überarbeitet werden)