

8 Dateikommandos

Ant kann mehr, als nur den Quellcode einer Anwendung zu kompilieren. Auch die Vor- und Nachbereitung, das Deployment oder sogar die Installation einer Anwendung ist möglich. Dazu muss Ant natürlich mit Dateien umgehen können. Als erste echte Kommandogruppe sollen daher die wichtigsten Tasks zur Dateiarbeit beschrieben werden.

8.1 Kopieren

Das wichtigste und am häufigsten benutzte Kommando aus der Reihe der Dateikommandos ist das `<copy>`-Kommando. Es dient zum Kopieren von Dateien und Verzeichnissen, wobei durch die Wirkung diverser Attribute, eingebetteter Tags und weiterer Elemente auch zusätzliche Manipulationen, z.B. an den Dateinamen, dem Dateiinhalt oder der Verzeichnisstruktur, möglich sind.

Das `<copy>`-Kommando kopiert Dateien und Verzeichnisse. Die Dateizugriffsrechte der Quelldatei werden beim Kopieren nicht übernommen. Vielmehr erhalten die neuen Dateien Standardattribute, die vom aktuellen Betriebssystem und den Rechten des Users abhängen (unter Unix z.B. von der `UMASK`-Einstellung). Das kann insbesondere unter Unix-Betriebssystemen zu Problemen führen, da natürlich auch ein eventuell vorhandenes Executable-Attribut verloren geht. Ausführbaren Programmen müssen Sie nach dem Kopieren entweder explizit neue Dateiattribute setzen oder Sie kopieren mit dem `cp`-Kommando des Betriebssystems. Hinweise dazu finden Sie im Abschnitt 17.2, der sich mit dem Aufruf von Systemkommandos beschäftigt.

Beim Kopieren sind zunächst natürlich Quell- und Zieldateien bzw. Verzeichnisse anzugeben. Dazu kommen wiederum unterschiedliche Methoden zum Einsatz. Die Angabe des Ziels hängt davon ab, ob eine einzelne, explizit benannte Datei oder möglicherweise mehrere Dateien und Verzeichnisse kopiert werden sollen.

Eine einzelne zu kopierende Datei kann durch das Attribut `file` festgelegt werden. Dieses Attribut führt keine Mustersuche durch. Wildcards werden nicht interpretiert, sondern als normales Zeichen gewertet. Die angegebene Datei muss

existieren, ansonsten erzeugt Ant einen Build-Fehler. Dieser kann allerdings durch das Attribut

```
failonerror="false"
```

abgeschaltet werden. In diesem Fall erzeugt Ant zwar eine Warnung, bricht den Build aber nicht ab.

Für die Vorgabe des Ziels stehen die alternativen Attribute `tofile` oder `todir` zur Verfügung. Eines der beiden Attribute muss im `<copy>`-Tag enthalten sein. Mit `tofile` können Sie einen kompletten Dateinamen einschließlich Pfad angeben, wobei der Name der Zieldatei natürlich vom Namen der Quelldatei abweichen kann. Mit `todir` wird ein Zielverzeichnis vorgegeben, in das die betreffende Datei kopiert wird.

Falls die Zieldatei bereits existiert, hängt es von verschiedenen Rahmenbedingungen ab, ob sie überschrieben wird oder nicht. Dazu später mehr. Hier zunächst zwei Beispiele, die das Kopieren einer einzelnen Datei demonstrieren:

```
<copy file="init.properties"
      tofile="server/startup.properties"/>
<copy file="init.properties"
      todir="server"/>
```

Um mehrere Dateien oder Verzeichnisse zu kopieren, muss das `<copy>`-Tag durch ein oder mehrere eingebettete `<fileset>`-Tags ergänzt werden. Auch in diesem Fall können sowohl das `tofile`- als auch das `todir`-Attribut zur Zielvorgabe benutzt werden. Allerdings erzeugt Ant bei Verwendung von `tofile` einen Build-Fehler, wenn durch das `<fileset>`-Tag mehr als eine Datei ausgewählt wird. Ein Fehler entsteht auch dann, wenn das im `Fileset`-Tag eingetragene Wurzelverzeichnis nicht existiert. Dieser Fehler kann ab der Ant-Version 1.6 ebenfalls durch das Attribut `failonerror` unterdrückt werden. Nachfolgend ein kleines Beispiel:

```
<copy todir="install">
  <fileset dir="build" includes="*.class"/>
  <fileset dir="etc">
    <include name="*.properties"/>
    <include name="*.resources"/>
  </fileset>
</copy>
```

Beim Kopieren von mehrstufigen Verzeichnissen wird die Verzeichnisstruktur unterhalb des Startverzeichnisses mitkopiert. Auch leere Unterverzeichnisse werden kopiert, wenn deren Namen zum Selektionsmuster der `Filesets` passen. Falls Sie das Kopieren leerer Verzeichnisse unterbinden möchten, können Sie dies durch das Attribut

```
includeemptydirs="false"
```

erreichen. Des Weiteren bewirkt

```
flatten="true"
```

dass die zu kopierenden Dateien direkt im Zielverzeichnis abgelegt werden, ohne die Verzeichnisstruktur der Quelle nachzubilden. Sollten gleichnamige Dateien existieren, wird eine bereits existierende Datei überschrieben.

Beim Kopiervorgang können so genannte Filter wirksam werden, die den Inhalt der zu kopierenden Dateien modifizieren. Meist handelt es sich dabei um das Ersetzen diverser Token durch andere Informationen, beispielsweise durch einen Copyright-Eintrag oder das Build-Datum. Nähere Informationen zu Filtern finden Sie im Kapitel 14 (Filter-Chains und Filter-Reader). Filter können sowohl außerhalb von `<copy>`-Kommandos als auch in Form eingebetteter Tags innerhalb des `<copy>`-Tags existieren. Damit extern definierte Filter wirksam werden, müssen sie durch das Attribut

```
filtering="true"
```

innerhalb eines `<copy>`-Kommandos freigeschaltet werden. Es werden dadurch alle globalen Filter wirksam; eine Auswahl spezieller Filter ist nicht möglich.

Das `<copy>`-Kommando arbeitet selektiv. Im Standardfall wird eine Datei nur dann kopiert, wenn die Zieldatei nicht existiert oder älter als die Quelldatei ist. Nur durch das Attribut

```
overwrite="true"
```

können Sie erzwingen, dass stets alle Dateien kopiert werden, unabhängig davon, ob die Zieldatei älter als die Quelldatei ist oder nicht.

Unter Microsoft-Betriebssystemen bleibt beim Überschreiben einer Datei deren Schreibweise (Groß-/Kleinschreibung) erhalten, auch wenn die Schreibweise der zu kopierenden Datei davon abweicht.

Einige andere Kommandos, z.B. das Kompilierkommando `<javac>`, berücksichtigen ebenfalls das Modifikationsdatum der Dateien, um unnötige Arbeitsschritte zu vermeiden. Beim Kopieren erhalten die kopierten Dateien aber die aktuelle Systemzeit als Modifikationsdatum. Sie würden damit auf jeden Fall als neu gewertet, auch wenn die Ursprungsdatei keiner echten Änderung unterlag. Durch das Attribut

```
preserveLastModified="true"
```

können Sie erzwingen, dass die Zieldatei dasselbe Modifikationsdatum erhält wie die Quelldatei.

Bei der betriebssystemübergreifenden Arbeit mit Textdateien entstehen gelegentlich Probleme durch unterschiedliche Zeichensätze. Das `<copy>`-Kommando kann daher beim Kopieren Konvertierungen vornehmen. Diese Konvertierungen steuern Sie durch die Attribute `encoding` und `outputencoding`. Die Encoding-Einstellungen werden wirksam, wenn die Datei gelesen wird. Das ist immer dann der Fall, wenn Filter zum Einsatz kommen. Das Output-Encoding hingegen wird beim Schreiben der Datei ins Zielverzeichnis benutzt.

Tabelle 8-1 zeigt Ihnen die Attribute des `<copy>`-Kommandos.

<copy>			
Attribut	Beschreibung	Default	Erforderlich
file	Name der zu kopierenden Datei. Keine Auswertung von Musterzeichen.		Entweder Attribut file oder eingebettetes Fileset
preservelastmodified	Die Kopien erhalten dasselbe Modifikationsdatum wie das Original.	false	Nein
tofile	Name der Zieldatei		Entweder tofile oder todir
todir	Name des Zielverzeichnisses		Entweder tofile oder todir
overwrite	Existierende Dateien werden auch dann überschrieben, wenn sie jünger sind als die Quelldatei.	false	Nein
filtering	Globale Filter berücksichtigen	false	Nein
flatten	Verzeichnisstruktur beim Ablegen im Zielverzeichnis ignorieren	false	Nein
include-emptydirs	Leere Verzeichnisse mitkopieren	true	Nein
failonerror	Build-Fehler erzeugen, wenn eine zu kopierende Datei nicht existiert	true	Nein
encoding	Zeichensatz-Kodierung beim Lesen (beim Einsatz von Filtern)		Nein
outputencoding	Zeichensatz-Kodierung beim Schreiben		Nein
enablemultiple-mappings	Beim Einsatz eines Mappers alle Transformationen berücksichtigen	false	Nein
verbose	Die Namen der kopierten Dateien auf der Konsole auflisten	false	Nein

Tab. 8-1 Attribute des <copy>-Kommandos

Das <copy>-Kommando akzeptiert eine überschaubare Menge von eingebetteten Tags. Bereits erwähnt wurde <fileset>. Ein weiteres, eng mit der Aufgabe des <copy>-Kommandos verbundenes Tag ist das <mapper>-Tag. Es ermöglicht die Modifikation von Dateinamen gemäß einer Transformationsvorschrift. Dieses Kommando ist sehr hilfreich, um bei Massen-Kopiervorgängen Datei- oder Verzeichnisnamen automatisch anzupassen. Sie können beispielsweise die Dateinamendung austauschen. Die diversen Mapper wurden bereits im Kapitel 7 beschrieben. Das folgende Beispiel würde beispielsweise dafür sorgen, dass alle Java-Dateien in ein Sicherungsverzeichnis kopiert werden und die Endung »bak« erhalten.

```
<copy todir="backup">
  <fileset dir="source" includes="*.java"/>
  <mapper type="glob" from="*.java" to="*.java.bak"/>
</copy>
```

Die beiden Tags `<filterchain>` und `<filterset>` definieren so genannte Filter. Diese Filter werden beim Kopieren wirksam. Sie modifizieren nach unterschiedlichen Regeln den Inhalt der zu kopierenden Dateien. Mögliche Funktionen sind das Ersetzen von Token, das Löschen von Kommentaren u.Ä.

In Tabelle 8–2 finden Sie nochmals alle von `<copy>` akzeptierten eingebetteten Tags.

<code><copy></code>		
Eingebettetes Tag	Aufgabe	Detaillierte Beschreibung in Kapitel
<code>filterchain</code>	Gruppe von Filterdefinitionen, mit denen die Inhalte der zu kopierenden Dateien beeinflusst werden können	14
<code>fileset</code>	Auswahl der zu kopierenden Dateien und Verzeichnisse	6.3
<code>filterset</code>	Weitere Filterdefinitionen zum Bearbeiten von Dateiinhalten	13.2
<code>mapper</code>	Element zur Umwandlung der Dateinamen	7

Tab. 8–2 Eingebettete Tags für das `<copy>`-Kommando

8.2 Verschieben

Das `<move>`-Kommando arbeitet ähnlich wie das `<copy>`-Kommando. Ein wesentlicher Unterschied besteht aber in der Defaulteinstellung des Attributs `overwrite`. Im Gegensatz zum `<copy>`-Kommando ist der Defaultwert hier `true`. Bereits existierende Dateien werden ohne weiteres überschrieben, auch wenn sie aktueller sind als die zu verschiebende Datei.

8.3 Verzeichnisse synchronisieren

Das `<sync>`-Kommando synchronisiert den Inhalt des Zielverzeichnisses mit dem Inhalt eines oder mehrerer Quellverzeichnisse, genauer gesagt, mit dem Inhalt eines oder mehrerer Filesets. Im Prinzip arbeitet es ähnlich wie das `<copy>`-Kommando. In einem ersten Arbeitsschritt werden alle Quelldateien, die aktueller sind als die entsprechenden Dateien im Zielverzeichnis, in dieses kopiert. In einem zweiten Arbeitsschritt löscht dieses Kommando alle Dateien aus dem Zielverzeichnis, die nicht auch in einem der Quellverzeichnisse existieren.

Als eingebettetes Tag akzeptiert dieses Kommando nur Filesets. Die verfügbaren Attribute zeigt Tabelle 8–3.

<sync>			
Attribut	Beschreibung	Default	Erforderlich
todir	Das Zielverzeichnis		Ja
overwrite	Existierende Dateien auch dann überschreiben, wenn sie aktueller sind	false	Nein
includeemptydirs	Leere Unterverzeichnisse ebenfalls kopieren	true	Nein
failonerror	Build abbrechen, wenn das Wurzelverzeichnis eines Filesets nicht existiert	true	Nein
verbose	Auflisten der kopierten Dateien	false	Nein

Tab. 8-3 Attribute des <sync>-Kommandos

8.4 Löschen

Mit <delete> können Sie Dateien und Verzeichnisse löschen. Für das gezielte Löschen einer Datei oder eines Verzeichnisses dienen die Attribute `file` und `dir`. Beide Angaben sind unabhängig voneinander. Das Attribut `dir` legt also nicht fest, in welchem Verzeichnis `file` gesucht wird, sondern welches Verzeichnis gelöscht werden soll. Das mittels `dir` festgelegte Verzeichnis wird komplett, also mit allen Unterverzeichnissen und Dateien, gelöscht. Im Attribut `file` hingegen können Sie einen Dateinamen, gegebenenfalls mit Pfadangabe, eintragen. Fehlt in `file` oder in `dir` eine absolute Pfadangabe, wird das aktuelle Arbeitsverzeichnis als Ausgangspunkt benutzt.

Neben der Verwendung dieser beiden Attribute ist es möglich, die zu löschenden Dateien über eingebettete <fileset>-Tags zu definieren. Die beiden folgenden Kommandos sind somit funktional identisch:

```
<delete dir="build/classes" />
<delete includeemptydirs="true">
  <fileset dir="build/classes" />
</delete>
```

Bei Vorgabe der Dateien durch ein Fileset werden standardmäßig nur Dateien gelöscht, übrig bleibt der leere Verzeichnisbaum. Um auch die leeren Verzeichnisse zu löschen, verwenden Sie das Attribut

```
includeemptydirs="true"
```

Da auch beim Löschen die Dateiausschlüsse wirksam werden, empfiehlt sich meist die Verwendung des Attributs `defaultexcludes`. Direkt im <delete>-Tag ist es allerdings veraltet, so dass es in den Filesets gesetzt werden sollte. Das Attribut

```
defaultexcludes="false"
```

hebt die Sonderrolle der reservierten Dateinamensmuster auf und ermöglicht so das problemlose Löschen der Dateien. Dazu nachfolgend ein Fragment zur Demonstration:

```
<delete includeemptydirs="true">
  <fileset dir="source" defaultexcludes="false"/>
</delete>
```

In Tabelle 8–4 sind die Attribute des `<delete>`-Kommandos aufgeführt.

<delete>			
Attribut	Beschreibung	Default	Erforderlich
file	Die zu löschende Datei		Entweder file oder dir oder ein eingebettetes Fileset
dir	Das zu löschende Verzeichnis		Entweder file oder dir oder ein eingebettetes Fileset
verbose	Die Namen der gelöschten Objekte auf der Konsole ausgeben	false	Nein
quiet	Fehlermeldungen ausgeben. Wenn auf true gesetzt, wird failonerror automatisch auf false gesetzt.	false	Nein
failonerror	Build-Fehler erzeugen, wenn ein Fehler beim Löschen auftritt. Dieses Attribut wird nur berücksichtigt bei quiet=false.	true	Nein
include-emptydirs	Leere Verzeichnisse löschen, wenn Filesets zur Dateiauswahl benutzt werden	false	Nein

Tab. 8–4 Attribute des `<delete>`-Kommandos

8.5 Verzeichnisse anlegen

Falls Sie manuell ein Verzeichnis anlegen müssen, steht dazu das Kommando `<mkdir>` zur Verfügung. Es enthält lediglich das unbedingt zu verwendende Attribut `dir`. Dieses Attribut nimmt genau einen exakt vorgegebenen Verzeichnisnamen auf.

Relative Pfadangaben beziehen sich immer auf das aktuelle Arbeitsverzeichnis der Build-Datei. Es kann daher sinnvoll sein, in einer Property-Datei eine absolute Wurzel anzugeben und diese im `<mkdir>`-Tag durch den variablen Teil des Pfadnamens zu ergänzen:

```
<mkdir dir="{pf.path.abs.build}/classes"/>
```

8.6 Abhängige Dateien löschen

Bestimmte Dateien können voneinander abhängig sein, ohne dass immer eine 1:1-Beziehung vorliegen muss. Als Beispiel sei nur ein Archiv genannt, dessen Aktualität vom Zustand mehrerer Dateien abhängt. Derartige Abhängigkeiten werden vom `<dependset>`-Tag ausgewertet. Letztendlich handelt es sich beim Kommando `<dependset>` um ein Löschkommando. Allerdings ist der Löschvorgang von Vorbedingungen abhängig. Innerhalb des Kommandos werden durch eingebettete Tags eine Liste mit Quell- und eine Liste mit Zieldateien definiert. Das Modifikationsdatum jeder Zieldatei wird mit dem Modifikationsdatum der Quelldateien verglichen. Ist eine der Quelldateien aktueller als eine der Zieldateien oder fehlt eine der Quelldateien, so werden alle Zieldateien gelöscht.

Sinnvoll ist der Einsatz eines Kommandos vor allem dann, wenn eine Zieldatei aus mehreren Quelldateien erstellt wird. Das kann z.B. der Fall sein, wenn eine HTML- oder JSP-Datei durch eine XSL-Transformation aus einer XML-Datei und einem Stylesheet erstellt werden soll.

Das Kommando `<dependset>` kennt vier eingebettete Tags. Es handelt sich dabei um `<srcfileset>`, `<srcfilelist>`, `<targetfileset>` und `<targetfilelist>`. Die Tags können mehrfach auftreten. Mit den beiden Source-Tags definieren Sie die Liste der Quelldateien, die beiden Target-Tags definieren die Menge der Zieldateien. Die Namen der vier Tags deuten darauf hin, dass es sich eigentlich um Varianten von `<fileset>` (siehe Abschnitt 6.3) bzw. `<filelist>` (siehe Abschnitt 6.5) handelt. Sie können exakt die Attribute und Sub-Tags besitzen, über die auch die beiden genannten allgemeinen Dateiselektions-Tags verfügen.

Einige Beispiele sollen das Kommando und seine Eigenschaften erläutern. Grundlage für das erste Beispiel sei folgendes Szenario: Für eine Anwendung werden Property-Dateien in einer XML-Datei gepflegt. Die Pflege erfolgt über einen XML-Editor, die Einträge in der XML-Datei werden mit einer passenden DTD validiert. Dies ermöglicht die Erstellung inhaltlich vollständiger und korrekter XML-Dateien und deren automatische Prüfung. Während des Build-Prozesses entsteht aus der XML-Datei eine der üblichen Java-Property-Dateien, z.B. durch eine XSL-Transformation. Um die Existenz ungültiger Property-Dateien zu verhindern, prüft das `<dependset>`-Kommando, ob die Property-Datei garantiert jünger ist als jede der drei Ausgangsdateien. Ist dies nicht der Fall, wird sie gelöscht.

```
<dependset>
  <srcfilelist dir="src/properties" files="serverprop.xml"/>
  <srcfilelist dir="src/properties" files="serverprop.dtd"/>
  <srcfilelist dir="tools" files="transformprops.xml"/>
  <targetfilelist dir="build/properties" files="server.properties"/>
</dependset>
```

Nicht immer kann die Menge der beteiligten Dateien explizit angegeben werden. Das zweite Beispiel löscht alle Dateien in einem Zielverzeichnis, wenn im Quell-

verzeichnis Java-Dateien enthalten sind, die jünger sind als eine der Zieldateien. Dieses Vorgehen ist zwar etwas rigoros, allerdings können Sie auf diese Weise die komplette Übersetzung eines Projekts erzwingen, wenn sich der Stand der Quelldateien verändert hat. Das ist nicht unbedingt notwendig, vermeidet aber Probleme durch class-Dateien, deren Quellen längst aus dem Projekt entfernt wurden.

```
<dependset>
  <srcfileset dir="source" includes="**/*" />
  <targetfileset dir="build/classes" includes="**/*" />
</dependset>
```

8.7 Dateiattribute manipulieren

Eine für Ant sehr wichtige Eigenschaft einer Datei ist das Modifikationsdatum. Viele Kommandos erstellen aus einer Quelldatei eine Zieldatei. Dabei wird die Aktion nur ausgeführt, wenn die Quelldatei jünger als die Zieldatei ist. Um definierte Zustände schaffen zu können, ist es oft notwendig, das Modifikationsdatum von Dateien auf einen bestimmten Wert zu setzen. Dies erfolgt mit dem Kommando `<touch>`. Dieses Kommando kann über das Attribut `file` entweder eine einzelne Datei oder mit einem eingebetteten Fileset eine Gruppe von Dateien und Verzeichnissen bearbeiten. Nachfolgend zwei Beispiele:

```
<touch file="bsp0606.xml" />

<touch>
  <fileset dir="Kapitel06">
    <include name="bsp*.xml" />
  </fileset>
</touch>
```

In der gezeigten Form erhalten die Dateien und Verzeichnisse die aktuelle Systemzeit als Modifikationszeitpunkt. Die beiden Attribute `millis` und `datetime` gestatten hingegen, auch einen anderen Zeitpunkt vorzugeben. Dabei bezeichnet `millis` einen Zeitpunkt, der um die angegebene Zahl von Millisekunden nach dem 1.1.1970 liegt. Im Attribut `datetime` können Sie den gewünschten Zeitpunkt in der Form `"MM/DD/YYYY HH:MM [AM | PM]"` vorgeben. Alle Bestandteile müssen enthalten sein. Andere Formate führen zu einem Build-Fehler.

```
<touch file="bsp0606.xml" datetime="09/01/2002 4:10 PM" />
```

Es existieren keine anderen, plattformunabhängigen Kommandos, um weitere Dateieigenschaften zu manipulieren. Das ist bedauerlich, da zumindest Zugriffsrechte auf Dateien und Verzeichnisse relativ häufig modifiziert werden müssen. Unter Unix-Systemen können Sie immerhin die Kommandos `<chmod>`, `<chown>` und `<chgrp>` verwenden, um die Unix-artigen Rechte zu setzen. Unter Windows-Systemen steht seit der Ant-Version 1.6 das Kommando `<attrib>` zur Verfügung.

Wegen der Bedeutung dieser Kommandos sollen die erwähnten Tasks trotz ihrer Plattformabhängigkeit hier beschrieben werden.

Das `<chmod>`-Kommando bildet als einziges der genannten vier Kommandos implizit ein Fileset nach. Das bedeutet, dass neben einem eingebetteten `<fileset>`-Tag auch alle Sub-Tags enthalten sein dürfen, die im `<fileset>`-Tag stehen können. Außerdem akzeptiert dieser Task natürlich auch eingebettete Filesets und Directory-Sets.

Die Liste der Attribute zeigt Tabelle 8-5. Einige der Attribute stammen aus dem `<fileset>`-Tag. Wirklich neu ist das Attribut `perm`, das eine Zeichenkette mit den zu setzenden Rechten aufnimmt. Diese entspricht in ihrem Aufbau den Attributen des Unix-`chmod`-Kommandos. Mit dem Attribut `type` können Sie festlegen, ob nur Dateien oder auch Verzeichnisse von den Änderungen betroffen sein sollen.

<code><chmod></code>			
Attribut	Beschreibung	Default	Erforderlich
<code>file</code>	Name einer Datei oder eines einzelnen Verzeichnisses, dessen Rechte geändert werden sollen		Entweder <code>file</code> oder <code>dir</code> oder eingebettete Filesets
<code>dir</code>	Name eines Verzeichnisses, in dem die Rechte aller enthaltenen Dateien geändert werden sollen		
<code>perm</code>	Die neuen Zugriffsrechte		Ja
<code>includes</code>	Die Liste aller einzuschließenden Dateien		Nein
<code>excludes</code>	Die Liste aller auszuschließenden Dateien		Nein
<code>default-excludes</code>	Liste mit Default-Ausschlüssen berücksichtigen	<code>yes</code>	Nein
<code>parallel</code>	Alle Änderungen auf Systemebene mit einem einzigen <code>chmod</code> -Aufruf ausführen	<code>true</code>	Nein
<code>maxparallel</code>	Anzahl der Dateien, die vom Systemkommando auf einmal modifiziert werden sollen (0 = keine Begrenzung)	0	Nein
<code>type</code>	Auswahl, ob nur Dateien, nur Verzeichnisse oder beides modifiziert werden soll. Mögliche Werte: <code>file</code> , <code>dir</code> und <code>both</code>	<code>file</code>	Nein
<code>verbose</code>	Statusmeldung ausgeben	<code>false</code>	Nein

Tab. 8-5 Attribute des `<chmod>`-Tasks

Mit dem Kommando `<chgrp>` können Sie die Gruppe, der eine Datei angehört, ändern. Dieser Task verwendet zur Dateiauswahl entweder das Attribut `file` oder aber die eingebetteten Tags `<fileset>`, `<filelist>` und `<dirset>`. Die übrigen Attribute (siehe Tabelle 8-6) entsprechen im Wesentlichen dem `<chmod>`-Task.

<chgrp>			
Attribut	Beschreibung	Default	Erforderlich
file	Name der Datei oder des Verzeichnisses		Ja, falls keine eingebetteten Tags verwendet werden
group	Name der neuen Gruppe		Ja
parallel	Alle Änderungen mit einem Systemaufruf ausführen	true	Nein
type	Auswahl, ob nur Dateien, nur Verzeichnisse oder beides modifiziert werden soll. Mögliche Werte sind: file, dir und both. Eingebettete Dirsets sind von dieser Einstellung nicht betroffen.	file	Nein
maxparallel	Anzahl der Dateien, die vom Systemkommando auf einmal modifiziert werden sollen (0 = keine Begrenzung)	0	Nein
verbose	Statusmeldung ausgeben	false	Nein

Tab. 8-6 Attribute des <chgrp>-Tasks

Der dritte Task, der nur unter Unix eingesetzt werden kann, ist der <chown>-Task. Er ändert den Eigentümer von Dateien. Die Funktionsweise entspricht dem <chgrp>-Task, lediglich das Attribut user tritt an Stelle des group-Attributs. Tabelle 8-7 zeigt die Übersicht über die Attribute des <chown>-Tasks.

<chown>			
Attribut	Beschreibung	Default	Erforderlich
file	Name der Datei oder des Verzeichnisses		Ja, falls keine eingebetteten Tags verwendet werden
user	Name des neuen Eigentümers		Ja
parallel	Alle Änderungen mit einem Systemaufruf ausführen	true	Nein
type	Auswahl, ob nur Dateien, nur Verzeichnisse oder beides modifiziert werden soll. Mögliche Werte sind: file, dir und both. Eingebettete Dirsets sind von dieser Einstellung nicht betroffen.	file	Nein
maxparallel	Anzahl der Dateien, die vom Systemkommando auf einmal modifiziert werden sollen (0 = keine Begrenzung).	0	Nein
verbose	Statusmeldung ausgeben	false	Nein

Tab. 8-7 Attribute des <chown>-Tasks

Die verschiedenen Windows-Versionen besitzen unterschiedliche Kommandos, um die Zugriffsrechte von Dateien zu ändern. Allen Versionen gemeinsam ist lediglich das Systemkommando ATTRIB, mit dem Sie aber nur sehr allgemeine Eigenschaften ändern können. Ant unterstützt momentan lediglich nur den Aufruf dieses einen Kommandos. Falls Sie z.B. ACLs ändern möchten, so müssen Sie die konkreten Systemkommandos aufrufen. Details dazu finden Sie in Abschnitt 17.2.

Der Task <attrib> ähnelt vom prinzipiellen Aufbau her den beiden Tasks <chgrp> und <chown>. Die zu bearbeitenden Dateien werden entweder durch das Attribut file oder eingebettete Filesets (<fileset>), Dateilisten (<filelist>) oder Verzeichnissesets (<dirset>) vorgegeben.

<attrib>			
Attribut	Beschreibung	Default	Erforderlich
file	Name der Datei oder des Verzeichnisses		Ja, falls keine eingebetteten Tags verwendet werden
readonly	Readonly-Attribut		Mindestens eines der nächsten vier Attribute
archive	Archiv-Attribut		
system	System-Attribut		
hidden	Hidden-Attribut		
parallel	Alle Änderungen mit einem Systemaufruf ausführen	true	Nein
type	Auswahl, ob nur Dateien, nur Verzeichnisse oder beides modifiziert werden soll. Mögliche Werte sind: file, dir und both. Eingebettete Dirsets sind von dieser Einstellung nicht betroffen.	file	Nein
maxparallel	Anzahl der Dateien, die vom Systemkommando auf einmal modifiziert werden sollen (0 = keine Begrenzung)	0	Nein
verbose	Statusmeldung ausgeben	false	Nein

Tab. 8-8 Attribute des <attrib>-Tasks