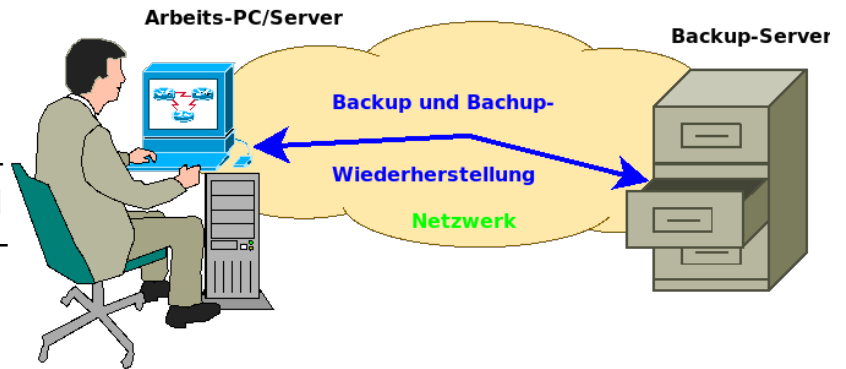


# Backup-Übung 01

**Aufgabe:** Verfasse ein Bash-Skript, welches mittels Dialog-Boxen den Ziel- und Quellpfad für ein Backup mittels „rsync“-Befehl erfragt! Dabei soll man unter mehreren nachfolgenden Optionen wählen können:

- nur neuere Dateien und Verzeichnisse kopieren.
- die neueren Dateien und Verzeichnisse auf den ursprünglichen Zustand wieder zurückführen.
- alle alten Dateien, die im Backup-Verzeichnis aber nicht im aktuellem Arbeitsverzeichnis vorhanden sind. Die Dateien die im Arbeitsverzeichnis in einer aktuelleren Version vorhanden sind, sollen in der älteren Version im Backupverzeichnis noch erhalten werden.
- Das Zielverzeichnis vorher reinigen und alles überschreiben.



*Bei der Anwendung von Dialog-Boxen tritt ein Problem auf. Der „rsync“-Befehl erwartet ein Passwort auf der Konsole!*

*Lösungshilfe zur Backupaufgabe*

1. Lösung: Verwende die „**sshpass**“-Erweiterung der Shell:

```
/usr/bin/rsync -ratlz --rsh="/usr/bin/sshpass -p 'password' ssh -o StrictHostKeyChecking=no -l 'username' " source_path destination_path
```

bzw. `sshpass -p "geheim" ssh fritz@pc1 'df -h'`

<https://www.tecmint.com/sshpass-non-interactive-ssh-login-shell-script-ssh-password/>

Alternativen:

2. Lösung: Verwende die EXPECT-Interpreter-Funktion:

```
#!/usr/bin/expect -f
spawn rsync -av /usr/local/sbin/hallo2 trebor@172.16.1.95:/home/trebor/hallo2
expect "password: "
send "robinhood031064\r"
```

3. Lösung: Verwende eine Passwortdatei:

```
rsync $args --password-file=rsync_pass user@rsynchoost::/share localdirectory
```

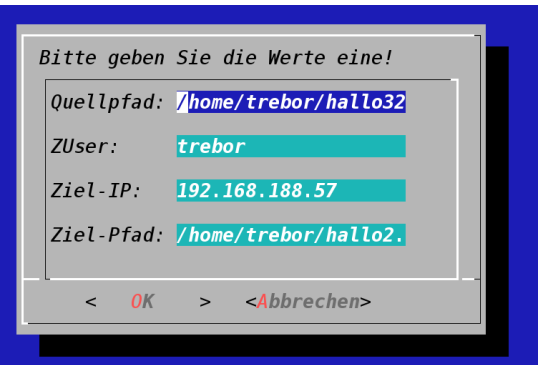
Weitere Hinweise finden Sie unter:

- <http://www.admin-magazine.com/Articles/Automating-with-Expect-Scripts>
- <https://unix.stackexchange.com/questions/111526/how-can-i-rsync-without-prompt-for-password-without-using-public-key-authentication>
- <https://www.linuxquestions.org/questions/linux-networking-3/automatically-pass-a-password-to-rsync-for-remote-host-882534/>
- <https://www.computerhope.com/unix/rsync.htm>

## 1. Ergänzung zur Lösung 1

Der Eingabe-Dialog  
mit Quell-, Zielpfad, Benutzerkennung und  
Ziel-IP-Adresse als Eingabeformular.

Hier die unsichere Teillösung mit  
sshpass:



Bitte geben Sie die Werte eine!

Quellpfad:

ZUser:

Ziel-IP:

Ziel-Pfad:

< OK > <Abbrechen>

```
#!/bin/bash
dialog --form "Bitte geben Sie die Werte eine!" 14 41 8 \
  "Quellpfad:" 1 1 "/home/trebor/hallo32" 1 12 20 38 \
  "ZUser:" 3 1 "trebor" 3 12 20 28 \
  "Ziel-IP:" 5 1 "192.168.188.57" 5 12 20 16 \
  "Ziel-Pfad:" 7 1 "/home/trebor/hallo2.txt" 7 12 20 38 2> \
  /tmp/hhh$$
Quelle="`head -n 1 /tmp/hhh$$`"
ZUser="`head -n 2 /tmp/hhh$$ | tail -n 1`"
Ziel="`head -n 3 /tmp/hhh$$ | tail -n 1`"
Zpfad="`tail -n 1 /tmp/hhh$$`"
rm /tmp/hhh$$
rsync -av -e "sshpass -p 'hallo123' ssh -p 22" -av \
  $Quelle $ZUser@$Ziel:$Zpfad
# Lösung: https://www.tecmint.com/sshpass-non-interactive-ssh-login-shell-
script-ssh-password/
```

## 2. Ergänzung zur Lösung 2:

Da der Befehle „expect“ ursprünglich nicht für die Bash-Konsole  
compiliert wurde, arbeiten expect-Befehle wie eigenständige shell-  
Anweisungen mit eigener Shebang!

Deshalb ist es sinnvoll die „expect“-Anweisungen in eine separate  
Skript-Datei auszulagern.

```
#!/bin/bash
echo „Bitte geben Sie Ihre Benutzerkennung an:“
read Benutzer
echo „Bitte geben Sie Ihr Passwort an:“
read Passwort
ein_expect_script $Benutzer $Passwort
```

Nebstehendes „bash“-Skript  
ruft das separat ausgearbeitete  
„ein\_expect\_script“ unterhalb  
auf.

Die beiden Werte  
(\$Benutzer und  
\$Passwort) werden als  
Übergabeparameter an  
das zweite Skript  
übergeben und dort als  
Tcl-Argumente (\$arg 0,  
bzw. \$arg1) übernommen  
und in die Variablen  
\$Ben und \$Pass übergeführt.

```
#!/usr/bin/expect -f
set Ben [lindex $argv 0];
set Pass [lindex $argv 1];
spawn rsync .... -e „ssh -p 22“ $Ben@host:/...
expect "Password: "
send "$Pass\r"
...
Text 1: ein_expect_script
```

## 2. Ergänzung zur Lösung 2 (Fortsetzung)

Eine weitere Alternative ist das Erstellen eines separaten „expect“-Skriptes durch das laufende bash-Skript selbst. Dieses wird gleich während der Ausführung des bash-Skriptes wiederum selbst zur Ausführung gebracht!

```
#!/bin/bash
pass="hallo123"
expectdatei="#!/usr/bin/expect -f
set timeout -1
spawn rsync -ravx /home/trebor/hallasdsado123/
trebor@192.168.188.57:/home/trebor/hallo1234/
expect \"password: \" { send \"$pass\\n\"}
expect \"#\"
"
# was geschieht in den beiden nachfolgenden Zielen?
echo "$expectdatei" > /tmp/temp.sh
chmod 777 /tmp/temp.sh

#exit # aktiviere diese Zeile und betrachte die „/tmp/temp.sh“-Datei!
/tmp/temp.sh
#echo "Fehler: `cat /tmp/fehler`"
if [[ $result -eq 0 ]]; then
    echo "Success"
else
    echo "Failure"
fi

rm -f /tmp/temp.sh
```

### **Achtung:**

Für die Anwendung von ssh- und rsync-Skripten gibt es noch weit bessere Lösungen.

Diese beruhen darauf, dass man eine Vertrauenssituation zwischen den Hosts (PCs) herstellt. Dies erfolgt mittels Key-Authentifizierung. Hierzu müssen bei asynchroner Verschlüsselung Private- und Public-Keys definiert und ausgetauscht werden oder nur ein symmetrischer Schlüssel bei symmetrischer Verschlüsselung.

... we could pass values (like 23 or 15) to the script from the command line.  
tclsh add.tcl 23 15

The method by which numbers can be passed into, and used by a script, is as follows.

**argc argv argv0**

All Tcl scripts have access to three predefined variables.

\$argc - number items of arguments passed to a script.

\$argv - list of the arguments.

\$argv0 - name of the script.

# Backup- oder FTP-Ersatz: Rsync hält Dateien aktuell

Von [Hans-Georg Esser](#) und [Heike Jurzik](#)

Quelle: [Aus Linux-Magazin 08/2007](#)

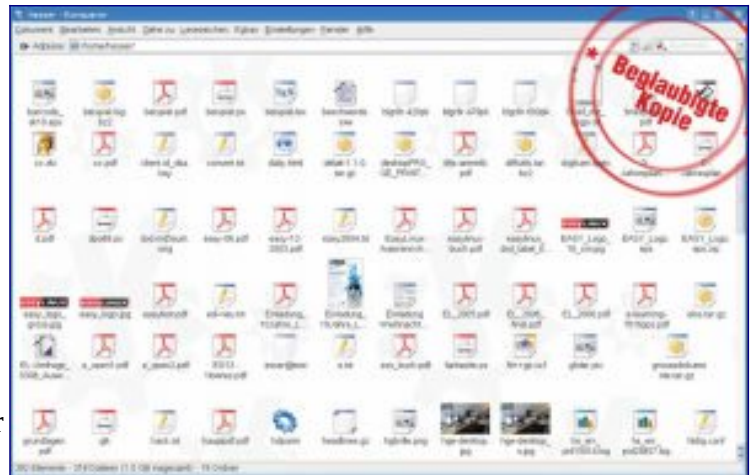
**Rsync ist der Klassiker für Synchronisierungsaufgaben wie Backup und Mirroring: Er aktualisiert Daten in einer Richtung, überträgt also geänderte Dateien von einem Master zur Kopie. Dieser Artikel stellt Rsync vor und verrät, worauf beim Syncen zu achten ist.**

Zwei Datenbestände zu synchronisieren, das bedeutet, sie identisch zu machen: Nach einer solchen Aktion liegen in den Verzeichnissen auf zwei Rechnern exakt die gleichen Dateien. Das ist in einigen Fällen eine leichte Aufgabe, etwa wenn einer der beiden Orte nur das Backup-Archiv ist, an dem es nie zu Änderungen kommt.

Anders sieht die Situation aus, wenn Anwender Dateien an einem Desktop-Rechner und parallel dazu auf einem Notebook bearbeiten, neu erzeugen oder löschen. Dann ist manchmal unklar, was zu tun ist, um einen synchronen Zustand zu erreichen.

In den kniffligeren Situationen geht es oft nicht ohne manuelles Eingreifen. Tools wie Unison (siehe Sysadmin-Einführung sowie [3], [6]) erkennen Problemkonstellationen und lassen die betroffenen Dateien in Ruhe; aus dem Log ergibt sich dann, auf welche Dateien der Anwender oder Administrator noch einen Blick werfen muss.

Ändern sich Daten immer nur an einer Stelle, reicht die einseitige Synchronisation aus. Die erledigt komfortabel ein klassisches, vom Samba-Team entwickeltes Tool namens Rsync [1], das lokal oder über das Netzwerk zwei Verzeichnishierarchien angleicht. Beim Transport übers Netz verwendet Rsync das identisch benannte eigene Netzwerkprotokoll (»rsync://«). Um den Abgleich so effizient wie möglich zu gestalten, zerlegt Rsync Dateien in Blöcke, vergleicht mit Hilfe von Prüfsummen und überträgt nur geänderte Teile.



## Datensicherung mit Rsync

Das Backup-Szenario ist wohl die populärste Anwendung von Rsync: Hier gibt es zu einem Verzeichnis an anderer Stelle eine exakte Spiegelung, die ein »rsync«-Aufruf regelmäßig oder nach größeren Änderungen aktualisiert. Modifikationen an den Dateien auf dem Backup-Rechner kommen dabei nicht vor, zur Synchronisation sucht Rsync also nur auf dem Quellrechner nach neuen, veränderten oder gelöschten Dateien.

Die einfachste Möglichkeit, dieses Verhalten zu erreichen, ist ein Befehlsaufruf der folgenden Form:

```
rsync -av --delete Quelle Ziel
```

Das Ziel ist dabei häufig ein Verzeichnis auf einem entfernten Rechner, für den Verbindungsaufbau verwendet Rsync standardmäßig eine SSH-Verbindung. (Ältere Versionen erfordern dafür noch die

Zusatzoption »-rsh=ssh«.) Dann hat das Ziel die Form »Rechner:Verzeichnis« oder »Benutzer@Rechner:Verzeichnis«. Ein vollständiges Beispiel sieht so aus:

```
rsync -av --delete /home/xyz/Daten test@backup.linux-magazin.de:/var/backup/xyz/
```

Die Option »-a« schaltet Rsync in den Archivmodus, sie ist eine Abkürzung für »-rlptgoD«: »-r« (rekursiv); »-lptgo« (symbolische Links, Rechte (permissions), Zugriffszeiten (times), Gruppe (group) und Besitzer (owner) bleiben erhalten; »-D« (Gerätedateien). Mit »-v« gibt sich das Tool nur etwas gesprächiger, »-delete« führt zum Löschen von Dateien im Backup, die es auf dem Quellrechner nicht länger gibt. Abbildung 1 zeigt einen Aufruf, der über die Option »-stats« noch mehr Informationen ausgibt.

## Sicher ist sicher

Die Option »-delete« ist mit Vorsicht zu genießen (siehe Kasten „Rsync-Fallen“). Es bietet sich daher an, mit dem Parameter »-n« einen Testlauf durchzuführen und in der Ausgabe zu überprüfen, welche Dateien Rsync im Ernstfall löscht. Eine andere Möglichkeit zur Sicherung der Daten auf dem Zielrechner ist die Option »-b«, die Löschkandidaten nicht einfach von der Platte fegt, sondern mit einem Backup-Suffix versieht. Standardmäßig nimmt Rsync dafür die Tilde; die Option »-suffix« setzt eine andere Erweiterung:

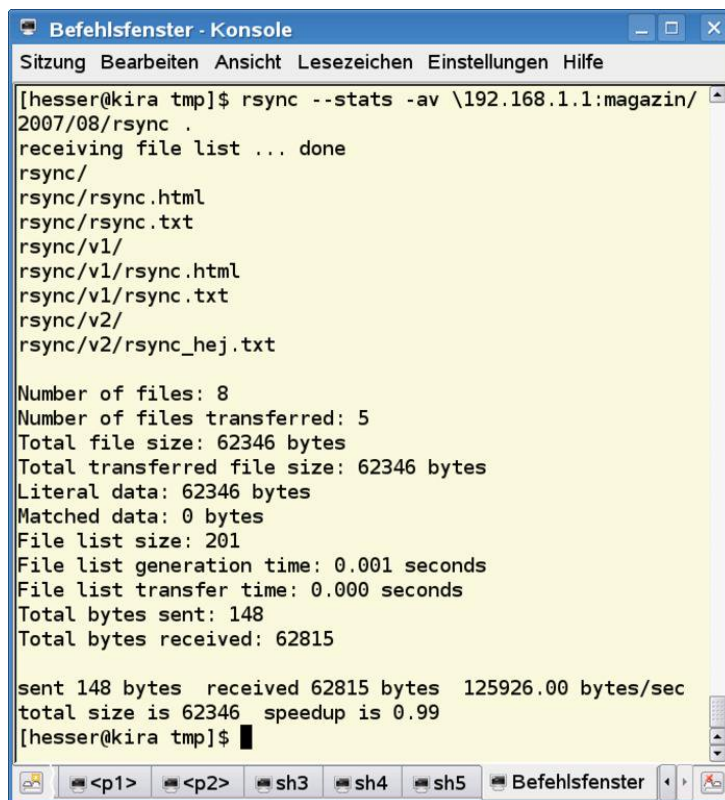
```
rsync -avb --suffix=.bak --delete  
/home/xyz/Daten test@backup.linux-  
magazin.de:/var/backup/xyz/
```

Darüber hinaus ist es möglich, die zu löschenden Dateien auf dem Zielrechner stattdessen in ein eigenes Verzeichnis zu sichern. Die Option »-backup-dir« akzeptiert wahlweise entweder einen Ordner relativ zum Homeverzeichnis (»-backup-dir=alt« schreibt die Sicherungskopien nach »~/alt« auf dem Zielrechner) oder eine absolute Pfadangabe.

Muss sich Rsync oft um den Abgleich zweier Maschinen kümmern, die beide veränderte Dateien enthalten, schützt die Option »-u« (update) vor Datenverlust. Sie verhindert, dass Rsync Dateien überschreibt, die auf dem Zielsystem einen neueren Zeitstempel haben als auf der Quelle. Dieser Mechanismus funktioniert nur korrekt, wenn beide Rechner mit derselben Systemzeit arbeiten, indem sie sich beispielsweise die Zeit von einem NTP-Server (Network Time Protocol) holen.

## Rsync tunen

Mehrere Optionen beschleunigen Rsync. So ist es beispielsweise möglich, Daten mit dem Parameter »-z« komprimiert auf die Reise zu schicken – ideal für langsame Verbindungen. Anders motiviert ist die Option



```
[hesser@kira tmp]$ rsync --stats -av \192.168.1.1:magazin/  
2007/08/rsync .  
receiving file list ... done  
rsync/  
rsync/rsync.html  
rsync/rsync.txt  
rsync/v1/  
rsync/v1/rsync.html  
rsync/v1/rsync.txt  
rsync/v2/  
rsync/v2/rsync_hej.txt  
  
Number of files: 8  
Number of files transferred: 5  
Total file size: 62346 bytes  
Total transferred file size: 62346 bytes  
Literal data: 62346 bytes  
Matched data: 0 bytes  
File list size: 201  
File list generation time: 0.001 seconds  
File list transfer time: 0.000 seconds  
Total bytes sent: 148  
Total bytes received: 62815  
  
sent 148 bytes  received 62815 bytes  125926.00 bytes/sec  
total size is 62346  speedup is 0.99  
[hesser@kira tmp]$
```

Abbildung 1: Höchst ausführlich: »rsync« gibt mit den Optionen »-v« und »-stats« nach den übertragenen Dateien noch statistische Daten aus.



»-bwlimit« (bandwidth limit). Hinter dem Parameter folgen ein Gleichheitszeichen und eine Angabe in Kilobit pro Sekunde, zum Beispiel:

```
rsync -avz --bwlimit=30 /home/xyz/Daten test@backup.linux-magazin.de:/var/backup/xyz/
```

Damit beschränkt Rsync freiwillig die von ihm genutzte Bandbreite, worüber sich alle übrigen Prozesse freuen, die dieselbe Datenleitung nutzen.

## Rsync in Skripten

Die bisher gezeigten Rsync-Aufrufe erfordern zur Authentifizierung jeweils die Eingabe des Benutzerpassworts, sind also interaktiv. Vollautomatische Backups mit Rsync-Aufrufen in Skripten sind aber auch möglich. Dazu ist es nur erforderlich, ein SSH-Schlüsselpaar zu erstellen und dem Zielrechner den öffentlichen Key zu übermitteln. Beim Erzeugen des Schlüsselpaars (»ssh-keygen -t dsa«) gibt der Anwender nur eine leere Passphrase ein, danach funktioniert die Anmeldung auf entfernten Rechnern ohne Kennwortabfrage.

Den öffentlichen Schlüssel (»~/.ssh/id\_dsa.pub«) trägt der Anwender dazu auf dem entfernten Rechner in die Datei »~/.ssh/authorized\_keys« ein. Sie bietet Platz für mehrere solcher Schlüssel, daher ist beim Kopieren des neuen Key darauf zu achten, keine bereits vorhandenen Einträge zu überschreiben. Praktischerweise liefert das OpenSSH-Paket das Tool »ssh-copy-id« mit, das einen Hostnamen als Argument erwartet und den Inhalt der öffentlichen Schlüsseldatei an die Datei »~/.ssh/authorized\_keys« auf dem entfernten Rechner anhängt:

```
$ ssh-copy-id -i ~/.ssh/id_dsa.pub remote.host
32
user@remote.host's password:
```

Um das Sicherheitsrisiko, das sich aus dem passwortlosen SSH-Schlüssel ergibt, einzuschränken, ist es möglich, für bestimmte Keys nur einzelne Kommandos zu erlauben. Dazu bearbeitet der Anwender auf dem Zielrechner die Datei »~/.ssh/authorized\_keys« mit einem Texteditor und ergänzt vor dem gewünschten Schlüssel (»ssh-dss ...«) einen Befehlseintrag in der Form »command="rsync Optionen"«. Dabei ist es wichtig, das erlaubte Kommando und den Schlüssel in eine Zeile ohne Umbruch zu schreiben.

## Teilabschnitte

Rsync eignet sich hervorragend dazu, große Datenmengen aktuell zu halten. Liegt auf der eigenen Platte beispielsweise das ISO-Image eines Release-Kandidaten für eine Linux-Distribution und ist diese nun erschienen, kann es den Download beschleunigen, »rsync« zum Aktualisieren der Imagedatei heranzuziehen statt das neue Image herunterzuladen, da sich vielleicht nur wenig verändert hat.

Das folgende Beispiel zeigt, wie Rsync ein lokales Debian-ISO-Image auf die Final-Version aktualisiert:

```
rsync -avz --progress
rsync://cdimage.debian.org/debian-cd/4.0_r0/i386/iso-dvd/debian-40r0-i386-DVD-1.iso
debian-testing-i386-DVD-1.iso
```

Bricht ein Transfer ab, löscht Rsync standardmäßig die schon übertragenen Teilstücke einer Datei. Die Option »-partial« verhindert dies und sorgt dafür, dass Rsync bei der Wiederaufnahme der Datenübertragung genau dort ansetzen und weitermachen kann.

Auf dem Zielrechner speichert Rsync die partielle Datei zunächst unter einem versteckten Dateinamen mit einer willkürlich gewählten Dateiendung (das gilt auch für Downloads ohne »-partial«; siehe Listing 3). Nach beendetem Transfer (oder auch bei einem Übertragungsabbruch) gibt Rsync der Datei ihren ursprünglichen Namen zurück (siehe auch Kasten „Rsync-Fallen“).

### Download vom Rsync-Server

```
01 $ # (in einem Terminal-Fenster den Download mit Rsync starten ...)
02 $ rsync -avz --partial --progress \
    rsync://cdimage.debian.org/debian-cd/4.0_r0/i386/iso-dvd/debian...-DVD-1.iso .
03 MOTD: Welcome to the rsync archive at Academic Computer Club, Umeå University.
04 receiving file list ...
05 1 file to consider
06 debian-40r0-i386-DVD-1.iso
07      1277952   0% 254.27kB/s   5:07:52
08
09 $ # (... und im anderen sehen, dass Rsync versteckte temporäre Dateien erzeugt)
10 $ ls -la
11 -rw----- 1 huhn huhn 3145728 2007-06-11 21:57 .debian-40r0-i386-DVD-1.iso.0lt27l
```

### Rsync-Fallen

Einige Rsync-Optionen sind unter Umständen dazu geeignet, großes Unheil hervorzurufen, wenn man sie unvorsichtig einsetzt. Wer die folgenden Standardfehler kennt, vermeidet Probleme.

- Der Klassiker: Die Option »-delete« ist mit Vorsicht zu genießen. Wer beispielsweise in einem Rsync-Aufruf versehentlich Quelle und Ziel vertauscht, löscht mit »-delete« zahlreiche Originaldateien. Um auf der sicheren Seite zu sein, empfiehlt sich ein Testlauf mit »-n«.
- Der abschließende Schrägstrich für Verzeichnisse sorgt ebenfalls oft für Verwirrung: Lautet der Aufruf zum Beispiel »rsync Optionen /home/xyz/Daten backup.linux-magazin.de:/var/backup/xyz/«, überträgt Rsync das Verzeichnis »Daten« zum Ziel, legt also »/var/backup/xyz/Daten« an. Steht im Aufruf hingegen »/home/xyz/Daten/«, sorgt der Schrägstrich dafür, dass nur der Inhalt von »Daten« übertragen wird. Eine Datei »~xyz/Daten/test.txt« wandert also gleich ins Zielverzeichnis und wird zu »/var/backup/xyz/test.txt« statt »../xyz/Daten/test.txt«.
- Abbruch der Datenübertragung bei Verwendung von »-partial«: Dass Rsync die bereits übertragenen Teilstücke unter dem Namen der Datei speichert, ist nicht immer vorteilhaft. Wenn der Anwender Rsync einsetzt, um eine bereits vorhandene größere Datei (beispielsweise ein ISO-Image) zu aktualisieren, und der Transfer abbricht, überschreibt Rsync das Original mit dem gerade übertragenen (kleineren) Teil. Um den Verlust des Originals zu verhindern, kann er vor dem Rsync-Aufruf einen Hardlink anlegen.
- Andere Server nicht überlasten: Es ist keine gute Idee, zwei Rsync-Instanzen auf dasselbe Verzeichnis loszulassen, da dies

den Load deutlich in die Höhe treiben kann. Startet ein Anwender Rsync beispielsweise via Cronjob, um regelmäßig einen Mirror zu aktualisieren, sollte er eine doppelte Skriptausführung über einen Locking-Mechanismus verhindern.

## Rsync-Daemon

Rsync gibt es auch als Server: Dazu startet der Systemadministrator den Daemon »rsyncd«, der wahlweise als selbstständiger Server läuft oder über passende Konfiguration von »inetd« beziehungsweise »xinetd« erst bei Kontaktaufnahme startet. Den Standard-Port 873 legt der Rsync-Eintrag in der Datei »/etc/services« fest. Um einen eigenständigen Rsync-Daemon zu aktivieren, ist nur eine Verlinkung des Startskripts aus »/etc/init.d/« in die richtigen Runlevel nötig, dann aktiviert das System »rsyncd« bei jedem Systemstart.

Auf Debian-Systemen ist es zusätzlich nötig, die Datei »/etc/default/rsync« anzupassen, damit das Startskript seine Arbeit aufnehmen kann. In »/etc/default/rsync« muss mindestens der Eintrag »RSYNC\_ENABLE=true« stehen. Rechner, auf denen noch der alte »inetd« läuft, überredet der Admin mit dem folgenden Eintrag in der Datei »/etc/inetd.conf« dazu, den Rsync-Daemon bei Bedarf zu aktivieren:

```
rsync stream tcp nowait root /usr/bin/rsync rsyncd --daemon
```

Für den moderneren »xinetd« leistet eine neue Datei »rsync« im Verzeichnis »/etc/xinetd.d/« das Gleiche (Listing 1). In beiden Fällen ist es auch möglich, statt »/usr/bin/rsync« den Daemon-Namen »/usr/sbin/rsyncd« anzugeben – beide Namen zeigen auf denselben Inode. Ein Vorteil des »xinetd« ist, dass er erlaubt, die Anzahl der Verbindungen pro IP-Adresse zu begrenzen – im Beispiel ist nur eine Verbindung pro IP-Adresse (»per\_source = 1«) zulässig.

**Listing 1:**  
**»/etc/xinetd.d/rsync«**

```
01 service rsync
02 {
03     socket_type  = stream
04     protocol     = tcp
05     wait         = no
06     user         = root
07     instances    = 50
08     per_source   = 1
09     server       = /usr/bin/nice
10     server_args  = /usr/bin/rsync --daemon
11 }
```

Unabhängig von der Startmethode liest der Rsync-Daemon die zentrale Konfigurationsdatei »/etc/rsyncd.conf«. Listing 2 zeigt im Abschnitt »[ftp]« ein Beispiel für Anonymous-Zugriff. Das Schlüsselwort »motd file« (message of the day) definiert die Begrüßungsmeldung des Servers, »uid« und »gid« legen User- und Gruppen-ID fest, mit denen der Daemon nach dem Verbindungsaufbau einen Kindprozess startet. Für Anonymous-Zugänge stehen hier oft »nobody« und »nogroup«. Die »max connections« legen fest, wie viele Clients sich insgesamt am Server anmelden dürfen.



**Listing 2:**  
**»/etc/rsyncd.conf«**

```
01 # GLOBAL OPTIONS
02
03 motd file = /etc/motd.rsync
04 uid = nobody
05 gid = nogroup
06 max connections = 50
07 socket options = SO_KEEPAIVE
08 timeout = 1200
09 log file = /var/log/rsyncd.log
10 transfer logging = true
11
12 # MODULE OPTIONS
13
14 [ftp]
15     comment = public archive
16     path = /var/www/pub
17     use chroot = yes
18     max connections=10
19
20 [debian-amd64]
21     comment = Debian Etch for AMD64
22     path = /home/ftp/pub/debian-amd64
23
24 [privater_Bereich]
25     comment = Meine eigenen Daten
26     path = /home/huhn
27     secrets file = /etc/rsyncd.huhn.secrets
28     auth users = huhn
29     uid = huhn
```

Um den Zugriff zu beschränken, bietet es sich an, Rsyncd über »xinetd« zu starten und dort die Anzahl der Verbindungen pro IP-Adresse einzuschränken; alternativ ist es möglich, die Zahl der maximalen Connects global oder für einzelne Freigaben zu beschneiden (Listing 2).

## Zugangskontrolle

Rsyncd erlaubt es dem Administrator auch, gezielt IP-Adressen oder ganze Bereiche in »/etc/rsyncd.conf« auszuklammern. Das geht mit dem Schlüsselwort »hosts deny«, hinter dem ein Gleichheitszeichen und dann eine Liste von IP-Adressen oder Hostnamen (durch Leerzeichen getrennt) folgen, zum Beispiel:

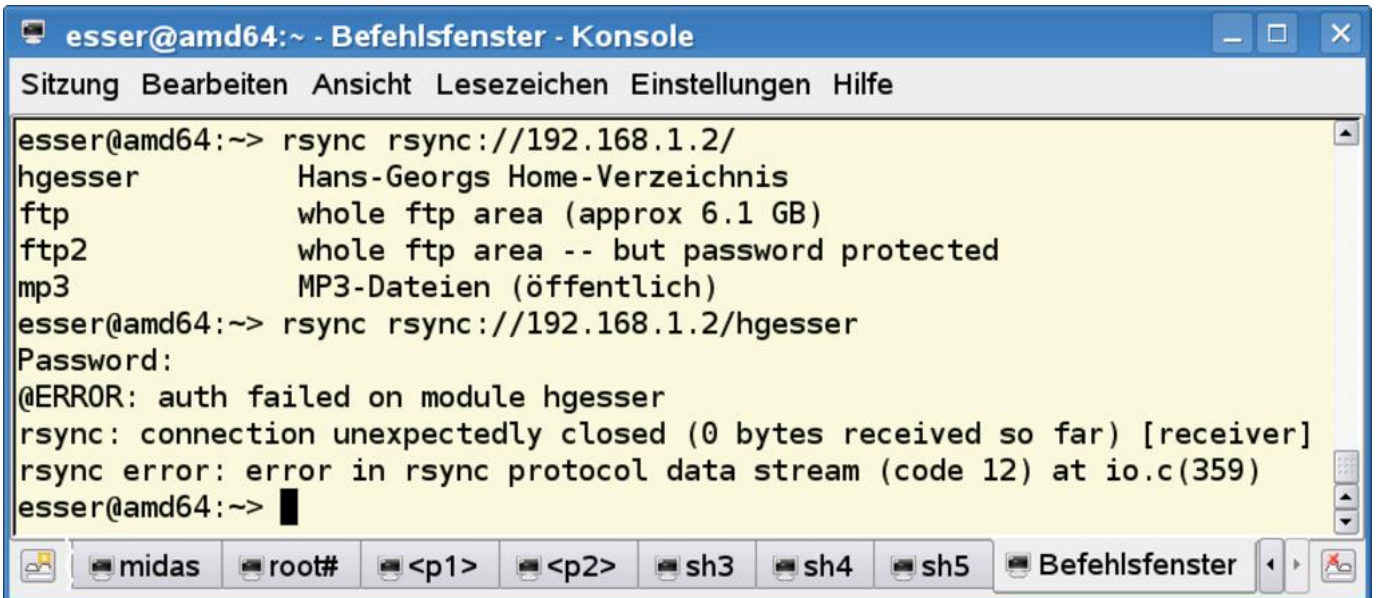
```
hosts deny = *.example.com 192.168.1.30 202.181.238.133/24
```

Listing 2 zeigt nicht nur einen Anonymous-Zugang, sondern auch die Freigabe »[privater\_Bereich]«, die durch ein Passwort geschützt ist und lediglich der Benutzerin »huhn« den Zutritt erlaubt. Im »secrets file« stehen der Benutzername und hinter dem Doppelpunkt das Kennwort im Klartext, etwa »huhn:sommer2007«. Da die Daten unverschlüsselt sind, ist es wichtig, diese Passwortdatei mit »chmod 600 /etc/rsyncd.huhn.secrets« zu verstecken – Rsync beschwert sich aussagekräftig im Logfile, wenn die Zugriffsrechte nicht passen (Listing 4).

Möchte ein Benutzer auf sein Homeverzeichnis zugreifen (das in der Regel nicht für alle lesbar ist), funktioniert das nur über passendes Setzen der »uid«. Die Benutzernamen im »secrets file« haben übrigens nichts mit den Systembenutzern aus »/etc/passwd« zu tun. Abbildung 2 zeigt, was ein Benutzer bei einem Passwortfehler zu sehen bekommt.

### Listing 3: Logfile-Auszüge

```
01 2007/06/11 13:32:46 [6668] rsyncd version 2.6.9 starting, listening on port 873
02 2007/06/11 13:32:55 [6681] connect from asteroid.huhnix.org (192.168.2.15)
03 2007/06/11 13:33:00 [6681] secrets file must not be other-accessible (see strict
modes option)
04 2007/06/11 13:33:00 [6681] continuing without secrets file
05 2007/06/11 13:33:00 [6681] auth failed on module privater_Bereich from
asteroid.huhnix.org (192.168.2.15): missing secret for user "huhn"
```



```
esser@amd64:~> rsync rsync://192.168.1.2/
hgesser          Hans-Georgs Home-Verzeichnis
ftp              whole ftp area (approx 6.1 GB)
ftp2             whole ftp area -- but password protected
mp3              MP3-Dateien (öffentlich)
esser@amd64:~> rsync rsync://192.168.1.2/hgesser
Password:
@ERROR: auth failed on module hgesser
rsync: connection unexpectedly closed (0 bytes received so far) [receiver]
rsync error: error in rsync protocol data stream (code 12) at io.c(359)
esser@amd64:~> █
```

Abbildung 2: Zugriff verwehrt: Wer das Passwort nicht kennt, erhält keinen Zugang zu Freigaben eines Rsync-Servers, die der Administrator mit einem »auth users«-Block geschützt hat.

## Backup mit Geschichte: Rsnapshot

Mit Rsync als Backup-Programm gibt es im Archiv immer genau eine Version – die letzte Sicherung. Dateien, die Anwender vor einem Rsync-Lauf löschen, verschwinden auch aus dem Backup. Das Tool Rsnapshot [2] schützt Daten etwas länger, indem es verschiedene Versionen der Verzeichnissammlung speichert. Damit kann der Admin auch auf den Hilferuf „Ich habe vorgestern eine Datei gelöscht“ souverän reagieren.

Er konfiguriert Rsnapshot über die Datei »/etc/rsnapshot.conf«, die unter anderem Informationen über die zu sichernden Verzeichnisse enthält. Gewöhnlich legt das Tool alle Backups im Verzeichnis »/.snapshots/« ab. Für regelmäßige Backups sorgen dann zwei Einträge in der Crontab, die »rsnapshot« mit dem Argument »hourly« oder »daily« aufrufen. Mit der Zeit entstehen so verschiedene Unterverzeichnisse »daily.0«, »daily.1« ... sowie »hourly.0«, »hourly.1« ..., die jeweils vollständige Backups enthalten. Unveränderte Dateien speichert Rsnapshot aber nicht redundant, sondern spart über Hardlinks Speicherplatz: Nur geänderte Dateien liegen dann in verschiedenen Versionen vor.

## Fazit

Rsync ist für alle Synchronisierungsaufgaben gut geeignet, die Dateien nur in einer Richtung aktualisieren; als Daemon kann es einen FTP-Server ersetzen. Für komplexere Szenarien, bei denen sich Dateien auf zwei Rechnern parallel verändern, ist aber ein Tool wie Unison besser geeignet.

## Infos

- [1] Rsync: [<http://samba.anu.edu.au/rsync/>]
- [2] Rsnapshot: [<http://www.rsnapshot.org/>]
- [3] Unison: [<http://www.cis.upenn.edu/~bcpierce/unison>]
- [4] Heike Jurzik, „Synchroner Datenstrom“: LinuxUser 04/06, S. 90 oder [<http://www.linux-user.de/ausgabe/2006/04/090-rsync/>]
- [5] Peer Heinlein, „Unwetterzentrale – Snapshot-Backups mit Rsync“: Linux-Magazin 09/04, S. 72 oder [[http://www.linux-magazin.de/heft\\_abo/ausgaben/2004/09/unwetterzentrale](http://www.linux-magazin.de/heft_abo/ausgaben/2004/09/unwetterzentrale)]
- [6] Daniel Rohark, „Datentandem – Daten synchronisieren mit Unison“: LinuxUser 02/04, S. 60 oder [<http://www.linux-user.de/ausgabe/2004/02/060-unison/>]

Dateien synchronisieren:

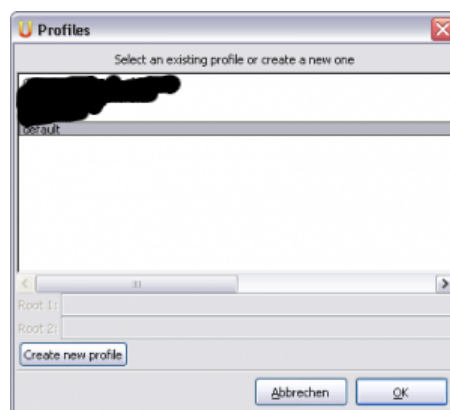
- \* mit FreeFileSync: <https://www.admin-magazin.de/Das-Heft/2017/02/Dateien-und-Ordnersynchronisation-mit-FreeFileSync>
- \* <https://www.admin-magazin.de/News/ownCloud-erhaelt-Delta-Sync-Funktion>
- \* mit unison: [https://www.admin-magazin.de/News/Tipps/Dateien-mit-Unison-synchronisieren?utm\\_source=Newsletter&utm\\_campaign=ADMIN-Newsletter-2017-20&utm\\_medium=email](https://www.admin-magazin.de/News/Tipps/Dateien-mit-Unison-synchronisieren?utm_source=Newsletter&utm_campaign=ADMIN-Newsletter-2017-20&utm_medium=email)

## Die **Windows – Linux – MAC – Kombilösung: Unison**




*Unison ist ein Programm für Windows, MAC und Linux zur Synchronisation von Dateien und Verzeichnissen auf verschiedenen Rechnern bzw. Festplatten/Datenträgern. Beim Abgleich werden nur Änderungen an den Daten synchronisiert, so dass ein erneuter Abgleich sehr schnell durchgeführt werden kann. Die größten Vorteile von Unison sind:*

- Das Programm läuft sowohl unter Linux als auch unter Windows. So lassen sich Verzeichnisse über Plattformen hinweg synchronisieren.
- Die Synchronisation funktioniert auch dann, wenn auf beiden Rechnern Dateien geändert wurden.
- Unison verwendet das rsync-Protokoll und hat eine (optionale) grafische Oberfläche.
- Eine schnelle Synchronisation über Netzwerke und das Internet ist möglich.

(aus [ubuntuusers.de](http://ubuntuusers.de))



Die Installation unter Windows ist ein bisschen aufwendiger als die Installation unter Linux. Neben Unison selber, dass von der Webpräsenz des Projekts geladen werden kann, sind noch weitere Programme notwendig.

-  [Unison für Windows](#)
  -  [sourceforge.net](http://sourceforge.net) Pidgin Projekt
  -  [Microsoft Visual C++ 2008 SP1 Redistributable Package \(x86\)](#)
1. Pidgin ist ein Instant Messenger, der es erlaubt, dasselbe Programm für verschiedene Instant-Messenger-Dienste zu nutzen. Ursprünglich wurde das Projekt unter dem Namen Gaim geführt, wegen markenrechtlicher Probleme jedoch in Pidgin umbenannt. Erweiterungen ermöglichen es, die jeweiligen Protokolle für Pidgin zu implementieren. (aus [ubuntuusers.de](http://ubuntuusers.de))  
Beim Download ist zu beachten, dass man die Pidgin-Version mit Gtk herunterlädt. Die Installation ist somit viel einfacher als wenn man Gtk einzeln downloadet. Nach der erfolgreichen Installation von Pidgin ist sollte auch Gtk auf dem Rechner installiert sein.
  2. Nach Pidgin ist das Microsoft Visual C++ 2008 SP1 Redistributable Package (x86) zu installieren. Die Installation sollte ohne Problem ablaufen.
  3. Nach der Installation von Pidgin, ist Gtk im Verzeichnis `C:\Programme\Gemeinsame Dateien\GTK\2.0\` zu finden. Damit Unison auch einwandfrei funktioniert, ist die .exe-Datei in das Verzeichnis `C:\Programme\Gemeinsame Dateien\GTK\2.0\bin` zu kopieren. Nach einem Doppelklick sollte Unison starten

Um die umständliche Installation auf Windows Systemen zu vermeiden, nehmen Sie einfach Unison Portable.



<http://www.portablefreeware.com/?id=979>