Versioner □□ (Git Alternative Lite)

Versioner, Debian (+Derivate)
Wiki-Stand: 14.06.2026
Script-Stand: 14.06.2026

Vorwort

Der **Versioner** ist ein Bash-Script zur automatisierten Dateiversionierung auf Linux-Systemen. Er ist als einfache, lokale und robuste Alternative zu Git gedacht, wenn Dateien automatisch überwacht und versioniert werden sollen, ohne dass hierfür ein vollständiges Git-Repository gepflegt werden muss.

Das Script eignet sich besonders für produktive Serverumgebungen, in denen regelmäßig Scripte, Konfigurationsdateien, PHP-Dateien, SQL-Dateien, Webdateien oder ähnliche Arbeitsdateien verändert werden. Ziel ist es, Änderungen nachvollziehbar zu speichern und frühere Versionen bei Bedarf wiederherstellen zu können.

Der Versioner arbeitet ohne Datenbank, ohne Dienst im Hintergrund und ohne komplexe Projektstruktur. Er wird gezielt aufgerufen, beispielsweise per Cron, scannt angegebene Ordner und speichert geänderte Dateien zentral unter:

```
/srv/.versioner
```

Dort entstehen pro überwachte Datei eigene Ablageordner mit Vollversionen, Patches, Indexdateien und Metadaten.

Der Versioner ist ausdrücklich kein vollständiger Ersatz für Git im Entwickler-Sinn. Er ist vielmehr ein praktischer Schutzmechanismus für Serverdateien, Scripte und Konfigurationsbestände.

⚙ Funktionen im Wesentlichen

Der Versioner durchsucht beim Scan die angegebenen Ordner rekursiv nach passenden Dateien. Dabei werden nur definierte Dateitypen berücksichtigt. Temporäre Dateien, Logs, Lockfiles und andere unerwünschte Dateitypen werden ausgeschlossen.

Wird eine Datei zum ersten Mal gefunden, speichert der Versioner eine komprimierte Vollversion. Bei späteren Änderungen wird nach Möglichkeit nur ein Patch gespeichert. Dadurch bleibt die Ablage kleiner und trotzdem nachvollziehbar.

Die wichtigsten Funktionen:

- rekursives Scannen frei angegebener Ordner
- mehrere Include-Ordner pro Aufruf möglich
- mehrere Exclude-Pfade pro Aufruf möglich
- feste Include-Dateitypen im Script
- feste Exclude-Dateitypen im Script
- Speicherung von Vollversionen
- Speicherung von Patches
- automatische Patch-Verifikation
- automatische Umwandlung in Vollversion, falls ein Patch nicht sauber verifizierbar ist
- Erkennung gelöschter Dateien
- Wiederherstellung alter Versionen
- Anzeige gespeicherter Versionen
- Prüfung bestehender Restore-Ketten
- Lockfile-Schutz gegen parallele Ausführung
- Schutz gegen gerade bearbeitete Dateien durch Stabilitätsprüfung

---

☝ Technische Hinweise

Der Versioner ist ein Bash-Script und benötigt eine typische Linux-Umgebung. Entwickelt wurde die Logik für Debian und Debian-Derivate.

Benötigte Programme:

```
bash
find
awk
stat
sha256sum
gzip
diff
patch
```

```
flock
mktemp
cp
```

In normalen Debian-Installationen sind diese Programme in der Regel bereits vorhanden oder über Standardpakete verfügbar.

Das Script arbeitet standardmäßig mit folgendem Basisordner:

```
/srv/.versioner
```

Darin werden die internen Daten abgelegt:

```
/srv/.versioner/files
/srv/.versioner/current
/srv/.versioner/state
/srv/.versioner/versioner.lock
```

Die überwachten Originaldateien bleiben an ihrem eigentlichen Ort. Der Versioner kopiert und versioniert sie nur zusätzlich.

Wichtig: Der Versioner sollte mit einem Benutzer ausgeführt werden, der die zu überwachenden Dateien lesen darf. Für Wiederherstellungen muss der Benutzer zusätzlich Schreibrechte auf die Zieldatei besitzen.

Grundprinzip

Der Versioner arbeitet nicht dauerhaft als Dienst, sondern wird bewusst gestartet.

Typischer Aufruf:

```
./versioner.sh scan -i "/srv/scripte" -i "/var/www/public" -i "/home/mariobeh/script-data" -e "/exec-log/" -e "/logs/" -e "/log/" -e "/cache/" -e "/tmp/"
```

Dabei bedeutet:

```
scan
```

Startet den Scan-Modus.

```
-i "/pfad"
```

Definiert einen Ordner, der rekursiv überwacht werden soll.

```
-e "/name/"
```

Definiert einen Pfadbestandteil, der ausgeschlossen werden soll.

Mehrere `-i` und mehrere `-e` sind erlaubt.

Mindestens ein `-i` ist Pflicht.

Speicherort und interne Struktur

Alle Versionierungsdaten werden unterhalb von:

```
/srv/.versioner
```

gespeichert.

Die wichtigsten Unterordner:

files

```
/srv/.versioner/files
```

Hier liegen die eigentlichen Versionen. Für jede überwachte Originaldatei wird eine parallele Ordnerstruktur angelegt.

Beispiel Originaldatei:

```
/srv/scripte/webcamloader.sh
```

Ablage im Versioner:

```
/srv/.versioner/files/srv/scripte/webcamloader.sh/
```

Darin befinden sich beispielsweise:

```
00001-1780574106.full.gz
00002-1780574166.patch.gz
00003-1780574286.patch.gz
index.tsv
meta
```

current

```
/srv/.versioner/current
```

Hier speichert der Versioner den zuletzt bekannten aktuellen Stand einer Datei. Dieser Stand wird als Vergleichsbasis für neue Patches genutzt.

Beispiel:

```
/srv/.versioner/current/srv/scripte/webcamloader.sh
```

state

```
/srv/.versioner/state
```

Dieser Ordner ist für Statusdaten vorgesehen.

Lockfile

```
/srv/.versioner/versioner.lock
```

Dieses Lockfile verhindert, dass der Versioner mehrfach gleichzeitig läuft.

Unterstützte Dateitypen

Der Versioner versioniert nicht pauschal jede Datei. Das ist bewusst so, damit keine Binärdateien, Logs, Cache-Dateien oder große Zufallsdaten in der Versionierung landen.

Berücksichtigt werden unter anderem:

```
*.sh
*.bash
*.zsh
*.py
*.php
*.js
*.css
*.html
*.htm
*.conf
*.cfg
*.ini
*.service
*.timer
*.txt
*.sql
*.json
*.xml
*.yml
*.yaml
*.md
```

Diese Include-Liste ist fest im Script definiert.

Ausgeschlossene Dateitypen

Bestimmte Dateitypen werden unabhängig vom Ordner ausgeschlossen.

Dazu gehören:

```
*.md5
*.state
*.pending
*.pid
*.lock
*.tmp
*.log
*.csv
```

```
cronlog.txt
```

Diese Dateien werden nicht versioniert.

Der Hintergrund ist einfach: Solche Dateien ändern sich häufig automatisch, enthalten oft Laufzeitdaten oder sind für eine Wiederherstellung als Quellversion nicht sinnvoll.

Scan-Modus

Der Scan-Modus ist die Hauptfunktion des Versioners.

Aufruf:

```
./versioner.sh scan -i DIR [-i DIR ...] [-e /DIR/ ...]
```

Beispiel:

```
./versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/mariobeh/script-data" \
  -e "/exec-log/" \
  -e "/logs/" \
  -e "/log/" \
  -e "/cache/" \
  -e "/tmp/"
```

Der Versioner durchsucht alle mit `-i` angegebenen Ordner rekursiv.

Dabei gilt:

- nicht vorhandene Include-Ordner werden übersprungen
- nur lesbare Dateien werden berücksichtigt
- ausgeschlossene Dateitypen werden übersprungen
- ausgeschlossene Pfadbestandteile werden übersprungen
- nur definierte Include-Dateitypen werden verarbeitet

Include-Ordner mit -i

Mit `-i` wird ein Startordner angegeben.

Beispiel:

```
-i "/srv/scripte"
```

Mehrere Include-Ordner sind möglich:

```
./versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data"
```

Mindestens ein `-i` ist Pflicht. Ohne Include-Ordner bricht der Versioner ab.

Beispiel falscher Aufruf:

```
./versioner.sh scan
```

Ergebnis:

```
Fehler: scan benötigt mindestens ein -i DIR
```

Exclude-Pfade mit -e

Mit `-e` werden Pfadbestandteile ausgeschlossen.

Beispiel:

```
-e "/cache/"
```

Das bedeutet: Sobald im Dateipfad der Bestandteil `/cache/` vorkommt, wird die Datei nicht versioniert.

Beispiele, die ausgeschlossen werden:

```
/var/www/public/cache/index.php
/var/www/public/projekt/cache/data.json
/srv/scripte/test/tmp/datei.sh
```

Beispiele, die nicht ausgeschlossen werden:

```
/var/www/public/cachebuster/index.php
/var/www/public/my-cache/index.php
/srv/scripte/blog/index.php
```

Die Slashes vorne und hinten sind wichtig, damit nur echte Pfadsegmente erkannt werden.

Empfohlene Excludes:

```
-e "/exec-log/"
-e "/logs/"
-e "/log/"
-e "/cache/"
-e "/tmp/"
```

Warum keine Standardordner?

Der Versioner soll bewusst keine festen Standardordner mehr im Script haben.

Das hat mehrere Vorteile:

- der Cronjob ist selbsterklärend
- die überwachten Ordner sind direkt im Aufruf sichtbar
- verschiedene Server können unterschiedliche Ordner nutzen
- ein versehentlicher Scan falscher Bereiche wird verhindert
- das Script bleibt allgemein verwendbar

Der Aufruf definiert vollständig, was überwacht wird.

Ablauf beim Scan

Beim Scan passiert grob Folgendes:

1. Der Versioner setzt ein Lockfile.
2. Die angegebenen Include-Ordner werden durchsucht.
3. Pro Kandidat werden Include- und Exclude-Regeln geprüft.
4. Von jeder gefundenen Datei werden Größe und Änderungszeit gespeichert.
5. Der Versioner wartet kurz.
6. Danach werden Größe und Änderungszeit erneut geprüft.

7. Nur stabile Dateien werden verarbeitet.
8. Neue Dateien werden als Vollversion gespeichert.
9. Geänderte Dateien werden als Patch oder Vollversion gespeichert.
10. Gelöschte Dateien werden im Index als gelöscht markiert.

Der kurze Warteabgleich ist wichtig, damit Dateien nicht mitten während einer Bearbeitung versioniert werden.

Schutz vor unfertigen Dateien

Ein zentrales Ziel des Versioners ist es, keine Dateien zu sichern, die gerade geschrieben oder bearbeitet werden.

Deshalb wird beim Scan eine Stabilitätsprüfung gemacht:

- Vor dem kurzen Warten wird Größe und Änderungszeit erfasst.
- Nach dem Warten wird beides erneut geprüft.
- Wenn sich etwas geändert hat, wird die Datei übersprungen.

Beispielmeldung:

```
/srv/scripte/test.sh → übersprungen, Datei änderte sich gerade
```

Zusätzlich erstellt der Versioner beim Speichern einen Snapshot per `cp -a`. Danach wird nochmals geprüft, ob Original und Snapshot identisch sind. Wenn nicht, wird die Datei ebenfalls übersprungen.

Vollversionen

Wenn eine Datei zum ersten Mal versioniert wird, speichert der Versioner eine Vollversion.

Beispiel:

```
00001-1780574106.full.gz
```

Eine Vollversion enthält den kompletten Dateiinhalt in gzip-komprimierter Form.

Vollversionen entstehen unter anderem bei:

- initialer Speicherung

- leerer Datei
- zu langer Patch-Kette
- zu großem Patch
- fehlgeschlagener Patch-Verifikation

Patches

Bei späteren Änderungen versucht der Versioner, nur einen Patch zu speichern.

Beispiel:

```
00002-1780574166.patch.gz
```

Ein Patch enthält nur die Unterschiede zwischen der zuletzt bekannten aktuellen Version und dem neuen Stand.

Das spart Speicherplatz und reicht für typische Script- oder Konfigurationsänderungen vollständig aus.

Patch-Ketten

Der Versioner begrenzt Patch-Ketten.

Standardwert:

```
MAX_PATCH_CHAIN=100
```

Nach 100 Patches wird wieder eine Vollversion gespeichert.

Der Grund: Sehr lange Patch-Ketten machen Wiederherstellungen aufwendiger und fehleranfälliger. Regelmäßige Vollversionen halten das System robust.

Maximale Patch-Größe

Der Versioner prüft, ob ein Patch im Verhältnis zur Quelldatei zu groß ist.

Standardwert:

```
MAX_PATCH_PERCENT=50
```

Wenn ein Patch größer als 50 Prozent der Quelldatei ist, wird stattdessen eine neue Vollversion gespeichert.

Beispielgrund im Index:

```
patch-too-large-78pct
```

Das verhindert unsinnige Patch-Dateien, wenn sich eine Datei stark geändert hat.

Indexdatei

Jede versionierte Datei erhält eine eigene Indexdatei:

```
index.tsv
```

Diese Datei enthält alle Versionen tabellarisch getrennt mit Tabs.

Typische Informationen:

- Versionsnummer
- Unix-Zeitstempel
- Typ
- Hash
- Dateirechte
- UID
- GID
- Dateigröße
- Artefaktdatei
- Grund

Beispiel sinngemäß:

```
00001  1780574106  full   HASH  755  1000  1000  12345  00001-1780574106.full.gz   initial
00002  1780574166  patch  HASH  755  1000  1000  12410  00002-1780574166.patch.gz  patch
```

Die Indexdatei ist die zentrale Grundlage für Anzeige, Restore und Verify.

Metadatei

Neben dem Index gibt es eine Metadatei:

```
meta
```

Darin steht unter anderem der ursprüngliche Pfad der Datei.

Beispiel:

```
path=/srv/scripte/webcamloader.sh
created=2026-06-14 03:20:00
```

Diese Information wird genutzt, um versionierte Dateien später wieder eindeutig zuordnen zu können.

Versionen anzeigen

Eine Datei kann ohne Unterbefehl abgefragt werden.

Aufruf:

```
./versioner.sh DATEI
```

Beispiel:

```
./versioner.sh webcamloader.sh
```

Dann zeigt der Versioner die letzten 5 Versionen dieser Datei.

Für mehr Versionen:

```
./versioner.sh webcamloader.sh 20
```

Dann werden die letzten 20 Versionen angezeigt.

Die Ausgabe enthält unter anderem:

- Versionsnummer
- Datum und Uhrzeit

- Typ
- Hash
- Dateigröße
- Grund

Beispiel:

```
v1 | 2026-06-14 03:10:00 | full  | HASH | 12345 Bytes | initial
v2 | 2026-06-14 03:15:00 | patch | HASH | 12410 Bytes | patch
```

Datei suchen

Bei der Anzeige oder Wiederherstellung kann entweder ein vollständiger Pfad oder ein Dateiname verwendet werden.

Beispiel mit Dateiname:

```
./versioner.sh webcamloader.sh
```

Beispiel mit vollständigem Pfad:

```
./versioner.sh /srv/scripte/webcamloader.sh
```

Wenn ein Dateiname mehrfach vorkommt, meldet der Versioner eine Mehrdeutigkeit und fordert den vollständigen Pfad.

Das ist wichtig, damit nicht versehentlich die falsche Datei wiederhergestellt wird.

Restore: Version wiederherstellen

Mit `restore` kann eine alte Version wiederhergestellt werden.

Aufruf:

```
./versioner.sh restore DATEI VERSION
```

Beispiel:

```
./versioner.sh restore webcamloader.sh 173
```

Vor der Wiederherstellung legt der Versioner ein Backup der aktuellen Datei an.

Beispiel:

```
webcamloader.sh.versioner-backup-20260614-032000
```

Danach wird die gewünschte Version rekonstruiert und an den Originalpfad geschrieben.

Dabei versucht der Versioner, ursprüngliche Rechte und Eigentümer wiederherzustellen:

```
chmod
chown
```

Falls `chown` mangels Rechte nicht möglich ist, wird dies ignoriert. Das Script bricht deswegen nicht ab.

Wiederherstellung gelöschter Zustände

Wenn eine Datei gelöscht wurde, markiert der Versioner dies im Index als:

```
deleted
```

Wird eine solche gelöschte Version wiederhergestellt, entfernt der Versioner die aktuelle Datei, falls sie existiert.

Auch hier wird vorher ein Backup erstellt, wenn die Datei aktuell vorhanden ist.

Verify: Restore-Ketten prüfen

Mit `verify` können alle aktuellen Restore-Ketten geprüft werden.

Aufruf:

```
./versioner.sh verify
```

Dabei versucht der Versioner, die jeweils letzte bekannte Version einer Datei aus den gespeicherten Vollversionen und Patches zu rekonstruieren. Anschließend wird das Ergebnis mit der aktuellen Originaldatei verglichen.

Wenn alles passt, passiert nichts weiter.

Wenn ein Patch defekt oder nicht sauber rekonstruierbar ist, reagiert der Versioner automatisch:

- bei defektem Patch wird der letzte Patch durch eine Vollversion ersetzt
- bei anderen Problemen wird eine Reparatur-Vollversion gespeichert

Am Ende erscheint:

```
Verify abgeschlossen
```

Gelöschte Dateien

Der Versioner erkennt gelöschte Dateien anhand vorhandener Metadaten.

Wenn eine Datei früher versioniert wurde, aber beim Scan nicht mehr existiert, wird im Index eine neue Version mit Typ `deleted` eingetragen.

Beispiel:

```
00008 1780575000 deleted - - - - 0 - deleted
```

Dadurch bleibt nachvollziehbar, wann eine Datei verschwunden ist.

Cron-Nutzung

Der Versioner ist besonders für Cron geeignet.

Beispiel für stündliche Ausführung:

```
0 * * * * /srv/scripte/versioner.sh scan -i "/srv/scripte" -i "/var/www/public" -i "/home/mariobeh/script-data" -e "/exec-log/" -e "/logs/" -e "/log/" -e "/cache/" -e "/tmp/" >> /var/log/versioner.log 2>&1
```

Beispiel für Ausführung alle 5 Minuten:

```
*/5 * * * * /srv/scripte/versioner.sh scan -i "/srv/scripte" -e "/tmp/" -e "/cache/" >> /var/log/versioner.log 2>&1
```

Der Lockfile-Schutz verhindert parallele Läufe. Wenn ein Lauf noch aktiv ist, bricht der nächste Lauf ab.

Beispielaufrufe

Einen Ordner scannen

```
./versioner.sh scan -i "/srv/scripte"
```

Mehrere Ordner scannen

```
./versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data"
```

Mehrere Ordner mit Ausschlüssen scannen

```
./versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data" \
  -e "/exec-log/" \
  -e "/logs/" \
  -e "/log/" \
  -e "/cache/" \
  -e "/tmp/"
```

Versionen einer Datei anzeigen

```
./versioner.sh webcamloader.sh
```

Mehr Versionen anzeigen

```
./versioner.sh webcamloader.sh 20
```

Alte Version wiederherstellen

```
./versioner.sh restore webcamloader.sh 173
```

Alle Restore-Ketten prüfen

```
./versioner.sh verify
```

Typische Einsatzbereiche

Der Versioner eignet sich besonders für:

- Bash-Scripte
- PHP-Projekte
- Konfigurationsdateien
- systemd-Units
- SQL-Dateien
- JSON/YAML/XML-Konfigurationen
- Markdown-Dokumentation
- Webdateien
- kleine bis mittelgroße Projektverzeichnisse

Weniger geeignet ist der Versioner für:

- große Binärdateien
- Mediendateien
- Datenbank-Dateien
- Cache-Verzeichnisse
- Log-Verzeichnisse
- Build-Artefakte
- node_modules

- vendor-Verzeichnisse, falls sehr groß

Solche Verzeichnisse sollten über `-e` ausgeschlossen werden.

Versioner vs. Git

Der Versioner ist keine vollständige Git-Alternative im Entwickler-Sinn.

Git bietet:

- Branches
- Commits mit Nachrichten
- Remote-Repositories
- Merge-Logik
- Zusammenarbeit mehrerer Personen
- Historienanalyse
- Staging
- Tags
- Pull/Push

Der Versioner bietet stattdessen:

- automatische Sicherung
- einfache Wiederherstellung
- keine Projektinitialisierung
- keine Git-Kenntnisse nötig
- keine manuelle Commit-Pflege
- einfache Nutzung per Cron
- zentrale Ablage auf dem Server

Der Versioner ist also weniger ein Entwicklerwerkzeug, sondern eher eine automatische Datei-Zeitmaschine für Serverdateien.

Sicherheit und Grenzen

Der Versioner schützt nicht vor allem.

Er schützt gut gegen:

- versehentliches Überschreiben

- versehentliche Scriptfehler
- kaputte Änderungen
- gelöschte Dateien
- unbemerkte kleine Änderungen
- fehlende manuelle Backups vor Bearbeitungen

Er schützt nicht vollständig gegen:

- Festplattenausfall
- Komplettverlust des Servers
- Angreifer mit Root-Zugriff
- mutwilliges Löschen von `/srv/.versioner`
- Datenbankverlust
- große Binärdatenänderungen
- fehlerhafte Systemzustände außerhalb der versionierten Dateien

Der Versioner ersetzt daher kein echtes Backup.

Empfehlung: `/srv/.versioner` selbst sollte regelmäßig extern gesichert werden.

Rechte und Eigentümer

Der Versioner speichert beim Versionieren:

- Dateimodus
- UID
- GID
- Dateigröße
- Hash

Beim Restore versucht er, Modus und Eigentümer wiederherzustellen.

Das ist besonders nützlich bei Scripten und Konfigurationsdateien.

Beispiel:

```
chmod 755 script.sh
chown user:gruppe script.sh
```

Wenn der Restore nicht als Root läuft, kann `chown` fehlschlagen. Das wird vom Script toleriert.

Hash-Prüfung

Jede Version wird mit einem SHA256-Hash gespeichert.

Beim Wiederaufbau einer Version wird der Hash geprüft. Wenn der rekonstruierte Inhalt nicht zum erwarteten Hash passt, bricht die Wiederherstellung ab.

Dadurch wird verhindert, dass beschädigte Patch-Ketten unbemerkt falsche Dateien erzeugen.

Warum Patches geprüft werden

Nach dem Speichern eines Patches baut der Versioner die neue Version testweise wieder zusammen. Anschließend wird geprüft, ob der rekonstruierte Inhalt dem Snapshot entspricht.

Wenn die Prüfung erfolgreich ist, wird der Patch akzeptiert.

Wenn die Prüfung fehlschlägt, wird der Patch verworfen und durch eine Vollversion ersetzt.

Das macht den Versioner robuster, weil fehlerhafte Patches nicht stillschweigend in der Historie bleiben.

Empfohlener Produktivaufruf

Für eine typische Serverumgebung:

```
/srv/scripte/versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data" \
  -e "/exec-log/" \
  -e "/logs/" \
  -e "/log/" \
  -e "/cache/" \
  -e "/tmp/"
```

Als Cronjob:

```
* * * * * /srv/scripte/versioner.sh scan -i "/srv/scripte" -i "/var/www/public" -i "/home/USER/script-data" -e "/exec-log/" -e "/logs/" -e "/log/" -e "/cache/" -e "/tmp/" >> /var/log/versioner.log 2>&1
```

Hinweise zur Ordnerauswahl

Nicht jeder Ordner sollte pauschal versioniert werden.

Sinnvoll:

```
/srv/scripte
/var/www/public
/home/USER/script-data
/etc
```

Mit Vorsicht:

```
/var/www
/home
/etc
```

Nicht sinnvoll:

```
/var/log
/tmp
/var/cache
/var/lib/mysql
/node_modules
/vendor
```

Je größer und dynamischer der Ordner, desto eher sollte er ausgeschlossen oder gar nicht erst als Include-Ordner gesetzt werden.

Fehlerbeispiele

Scan ohne -i

```
./versioner.sh scan
```

Fehler:

```
Fehler: scan benötigt mindestens ein -i DIR
```

Unbekannter Scan-Parameter

```
./versioner.sh scan -x test
```

Fehler:

```
Fehler: Unbekannter scan-Parameter: -x
```

Restore mit unklarem Dateinamen

Wenn mehrere Dateien denselben Namen haben:

```
./versioner.sh config.php
```

Dann meldet der Versioner eine Mehrdeutigkeit und verlangt den vollständigen Pfad.

Wartung

Der Versioner wächst mit der Anzahl der Änderungen.

Mögliche Wartungsmaßnahmen:

- gelegentlich Speicherplatz prüfen
- `/srv/.versioner` in Backups einbeziehen
- alte Versionen bei Bedarf manuell archivieren
- sehr große oder dynamische Ordner nicht scannen
- Exclude-Liste im Cron-Aufruf sauber pflegen
- regelmäßig `verify` ausführen

Beispiel:

```
./versioner.sh verify
```

Bekannte Grenzen

Der Versioner arbeitet dateibasiert. Er kennt keine semantischen Änderungen und keine Projektzusammenhänge wie Git.

Wenn zehn Dateien zusammen geändert wurden, weiß der Versioner nicht, dass diese Änderung logisch zusammengehört. Jede Datei hat ihre eigene Historie.

Auch gibt es keine Commit-Nachrichten. Der Grund einer Version ist technisch, beispielsweise:

```
initial
patch
patch-chain-limit
patch-too-large-80pct
deleted
```

Der Versioner ist bewusst einfach gehalten.

⚠ Haftungsausschluss

Der Versioner ist ein Hilfswerkzeug zur lokalen Dateiversionierung.

Er ersetzt kein vollständiges Backupkonzept.

Ich übernehme keine Haftung für:

- Datenverlust
- fehlerhafte Bedienung
- beschädigte Dateien
- falsche Restore-Versionen
- fehlende Backups
- falsch gesetzte Include- oder Exclude-Pfade
- zu große oder ungeeignete Verzeichnisbereiche
- Schäden durch produktiven Einsatz ohne vorherigen Test

Vor produktivem Einsatz sollte das Script in einer ungefährlichen Umgebung getestet werden.

Besonders Restore-Vorgänge sollten vorab verstanden und geprüft werden.

Script Download

Das Script kann hier in immer der neuesten Version heruntergeladen werden:

DOWNLOAD (via mariobeh.de)

Das Script sollte zentral abgelegt werden, zum Beispiel:

```
/srv/scripte/versioner.sh
```

Ausführbar machen:

```
chmod +x /srv/scripte/versioner.sh
```

Testaufruf:

```
/srv/scripte/versioner.sh help
```

Produktivaufruf:

```
/srv/scripte/versioner.sh scan \
  -i "/srv/scripte" \
  -i "/var/www/public" \
  -i "/home/USER/script-data" \
  -e "/exec-log/" \
  -e "/logs/" \
  -e "/log/" \
  -e "/cache/" \
  -e "/tmp/"
```

Nur in Deutsch verfügbar.

Vielen Dank,
mariobeh.

Revision #3
Created 14 June 2026 11:40:22 by Admin
Updated 14 June 2026 11:51:41 by Admin