# Firebird’s gbak Backup and Restore Utility

Norman Dunbar, Mark Rotteveel

Version 1.16, 24 February 2024

# Table of Contents

# Chapter 1. Introduction

`Gbak` is one of the database backup and restore utilities supplied with Firebird. In Firebird 1.5 and earlier, it is the only supplied utility of this kind, while Firebird 2.x and later also has the `nbackup` utility which is described in *Firebird's nbackup tool*.

In this manual, we will discuss:

- Command-line options for `gbak`.
- `gbak` commands and their parameters.
- Running `gbak` in backup or restore modes.
- Some caveats, gotchas and foibles of `gbak`.

# Chapter 2. Command-line Options

## 2.1. Common Options

When running `gbak` in backup or restore mode, there are a number of options which apply to either mode.

We describe them in the sections below.

### 2.1.1. -?

Displays the commandline options and switches.

*gbak Usage information for Firebird 5.0, with links to relevant sections*

```
gbak:Usage:
     gbak -b <db set> <backup set> [backup options] [general options]
     gbak -c <backup set> <db set> [restore options] [general options]
     <db set> = <database> | <db1 size1>...<dbN> (size in db pages)
     <backup set> = <backup> | <bk1 size1>...<bkN> (size in bytes = n[K|M|G])
     -recreate overwrite and -replace can be used instead of -c
gbak:legal switches are:
    -B(ACKUP_DATABASE)    backup database to file
    -C(REATE_DATABASE)    create database from backup file (restore)
    -R(ECREATE_DATABASE) [O(VERWRITE)] create (or replace if OVERWRITE used)
                                database from backup file (restore)
    -REP(LACE_DATABASE)   replace database from backup file (restore)
gbak:backup options are:
    -CO(NVERT)            backup external files as tables
    -E(XPAND)             no data compression
    -FA(CTOR)             blocking factor
    -G(ARBAGE_COLLECT)    inhibit garbage collection
    -IG(NORE)             ignore bad checksums
    -L(IMBO)              ignore transactions in limbo
    -NOD(BTRIGGERS)       do not run database triggers
    -NT                   Non-Transportable backup file format
    -OL(D_DESCRIPTIONS)   save old style metadata descriptions
    -T(RANSPORTABLE)      transportable backup -- data in XDR format
    -ZIP                  backup file is in zip compressed format
gbak:restore options are:
    -BU(FFERS)            override page buffers default
    -FIX_FSS_D(ATA)       fix malformed UNICODE_FSS data
    -FIX_FSS_M(ETADATA)   fix malformed UNICODE_FSS metadata
    -I(NACTIVE)           deactivate indexes during restore
    -K(ILL)               restore without creating shadows
    -MO(DE) <access>      "read_only" or "read_write" access
    -N(O_VALIDITY)        do not restore database validity conditions
    -O(NE_AT_A_TIME)      restore one table at a time
    -P(AGE_SIZE)          override default page size
```

```
    -REPLICA <mode>      "none", "read_only" or "read_write" replica mode
    -USE_(ALL_SPACE)      do not reserve space for record versions
gbak:general options are:
    -CRYPT                crypt plugin name
    -D(IRECT_IO)          direct IO for backup file(s)
    -FE(TCH_PASSWORD)     fetch password from file
    -KEYHOLDER            name of a key holder plugin
    -KEYNAME              name of a key to be used for encryption
    -M(ETA_DATA)          backup or restore metadata only
    -PAR(ALLEL)           parallel workers
    -PAS(SWORD)           Firebird password
    -RO(LE)               Firebird SQL role
    -SE(RVICE)            use services manager
    -SKIP_D(ATA)          skip data for table
    -INCLUDE(_DATA)       backup data of table(s)
    -ST(ATISTICS) TDRW    show statistics:
        T                 time from start
        D                 delta time
        R                 page reads
        W                 page writes
    -TRU(STED)            use trusted authentication
    -USER                 Firebird user name
    -V(ERIFY)             report each action taken
    -VERBI(NT) <n>        verbose information with explicit interval
    -Y  <path>            redirect/suppress status message output
    -Z                    print version number
gbak:switches can be abbreviated to the unparenthesized characters
```

The links are not present in the actual `gbak` output.

The parentheses shown in the above indicates how much of the switch name you need to use to avoid ambiguity. In this manual we indicate this with square brackets instead. Once you have specified the absolute minimum — the part before the opening '(' — or '[' — you can use as much of what follows as you wish. For example, to use the `-b[ackup_database]` switch the minimum you must supply on the command line is `-b` but anything between `-b` and `-backup_database` will be accepted.

The `-?` switch was introduced in Firebird 2.5, but older versions will also display the usage (together with an error) when an invalid switch is provided.

### 2.1.2. -CRYPT

Crypt plugin name.

*Syntax*

```
-CRYPT cryptplugin
```

The -CRYPT option will generally need to be combined with the -KEYHOLDER and -KEYNAME options.

On backup of a non-encrypted database, the -CRYPT option specifies the encryption plugin to use to encrypt the backup file.

When backing up an encrypted database, specifying -CRYPT is not necessary as by default it will use the same plugin and key as the database itself.

On restore of a non-encrypted backup, the -CRYPT option will encrypt the new database using the specified plugin.

It is not possible to backup an encrypted database to an unencrypted backup file, or to restore an encrypted backup file to an unencrypted database.

The database is first encrypted right after creation and only after the encryption configuration is set in the header. This is a bit faster than a “restore-then-encrypt” approach, but, mainly, it is to avoid having non-encrypted data on disk during the restore process.

Introduced in Firebird 4.0.

### 2.1.3. -D[IRECT_IO]

Direct IO for backup file(s).

When enabled, gbak creates — on backup, or opens — on restore, the backup file in direct IO (or unbuffered) mode. In this mode, the backup file is not cached by the filesystem cache.

Usually, a backup is written — on backup, or read — on restore, just once, and there is no real benefit from caching its content. Performance should not suffer as gbak uses sequential IO with relatively big chunks.

Direct IO mode is silently ignored if the backup file is written to the standard output, or read from the standard input (i.e. if the backup filename is stdout or stdin).

Added in Firebird 5.0.

### 2.1.4. -FE[TCH_PASSWORD]

Fetch password from file (or standard input).

*Syntax*

```
-FE[TCH_PASSWORD] { password-filename | stdin | /dev/tty }
```

This switch causes the password for the appropriate user to be read from a file as opposed to being specified on the command line. The filename supplied is *not* in quotes and must be readable by the user running gbak. If the filename is specified as stdin, then the user will be prompted for a password. On POSIX systems, the filename /dev/tty will also result in a prompt for the password.

Introduced in Firebird 2.5.

### 2.1.5. -KEYHOLDER

Name of a keyholder plugin

*Syntax*

```
-KEYHOLDER keyholder-name
```

The `-KEYHOLDER` option must be specified to backup an encrypted database, to create an encrypted backup of an unencrypted database, to restore an encrypted backup, or to restore an unencrypted backup to an encrypted database.

It is not possible to backup an encrypted database to an unencrypted backup file, or to restore an encrypted backup file to an unencrypted database.

Introduced in Firebird 4.0.

### 2.1.6. -KEYNAME

Name of the key to be used for encryption.

```
-KEYNAME key-name
```

The `-KEYNAME` option can be used to create an encrypted backup of an unencrypted database, to restore an encrypted backup with a non-default key name, or to restore an unencrypted database to an encrypted database.

This option must generally be combined with `-KEYHOLDER` and `-CRYPT`.

It is not possible to backup an encrypted database to an unencrypted backup file, or to restore an encrypted backup file to an unencrypted database.

Introduced in Firebird 4.0.

### 2.1.7. -M[ETA_DATA]

Perform metadata-only backup or restore.

This switch causes your data to be ignored and not backed up or restored. In a backup, only the database metadata (tables, triggers, etc.) are backed up. When used in a restore, only the database metadata are restored, and any data in the backup file will not be restored. This switch can be used when creating database clones which are required to contain only the tables, indices, etc., but no data.

### 2.1.8. -PAR[ALLEL]

Number of parallel workers to use during backup or restore.

*Syntax*

```
-PAR[ALLEL] worker-count
```

The default number of parallel workers is 1 (one), but — for restore — this is not identical to explicitly specifying `-PARALLEL 1`.

For backup, this option controls the number of connections used to read user-data. Every additional worker creates its own thread and connection to read data in parallel with other workers. All worker connections share the same database snapshot to ensure a consistent data view across all workers. Workers are created and managed by `gbak` itself. The database metadata is read by a single thread.

For restore, this option controls the number of connections used to write user-data, and to configure the number of parallel workers used for index creation. Every additional worker creates its own thread and connection to write data in parallel with other workers. The database metadata is still created using a single thread — the "main" connection.

This "main" connection uses DPB tag `isc_dpb_parallel_workers` to pass the value of switch `-PARALLEL` to the engine — to use the engine's ability to build indices in parallel. If the `-PARALLEL` switch is not specified, `gbak` will write data using a single thread and will not use DPB tag `isc_dpb_parallel_workers`. In this case, the engine will use the value of `ParallelWorkers` in `firebird.conf` when building indices, i.e. this phase could be run in parallel by the engine itself.

To fully avoid parallel operations when restoring a database, use `-PARALLEL 1`.

The `ParallelWorkers` and `MaxParallelWorkers` settings in `firebird.conf` have no effect on `gbak`, except during index creation. `MaxParallelWorkers` can limit the number of parallel workers during index creation, and `ParallelWorkers` is used for index creation if `-PARALLEL` is not specified.

Introduced in Firebird 5.0

### 2.1.9. -PAS[SWORD]

Password for authentication.

*Syntax*

```
-PAS[SWORD] password
```

This need not be supplied if `ISC_PASSWORD` environment variable exists and has the correct value.

### 2.1.10. -ROLE

Role name for privileges.

*Syntax*

```
-RO[LE] role-name
```

Allows the specification of the role to be used by the connecting user.

### 2.1.11. -SE[RVICE]

Perform backup or restore through service manager.

*Syntax*

```
-SE[RVICE] service-name
```

This switch causes `gbak` to backup or restore a remote database via the service manager. This causes the backup file to be created or read on the remote server, so the path format and filename must be valid on the remote server. For Firebird 3.0 and earlier, the servicename must always end in `service_mgr`.

More details on the syntax of *service-name* can be found in Remote Backups & Restores.

You can use this option to connect to a locally hosted database as well. If you do, taking a backup or restoring using this option can run quicker than accessing the database directly. See also Speeding up the Backup.

### 2.1.12. -SKIP_D[ATA]

Exclude table data from backup or restore for matching table names.

*Syntax*

```
-SKIP_D[ATA] sql-regex
```

The backup or restore skips the data for table(s) matching the SQL regular expression argument. Opposite of `-INCLUDE[_DATA]`. To skip all data, use `-M[ETA_DATA]`.

The metadata of the table *is* included, only their data is skipped.

Excluding data from a backup or restore can yield errors during restore when constraints are enabled, and a foreign key constraint exists on a table not excluded, depending on a table that was excluded.

You can use `-I[NACTIVE]` to disable **all** indices, primary key, unique key and foreign key constraints. If `CHECK` constraints depend on excluded data, you may also need

to specify `-N[O_VALIDITY]`.

The SQL regular expression syntax is documented in the *Firebird 5.0 Language Reference*.

Introduced in Firebird 3.0.

### 2.1.13. -INCLUDE[_DATA]

Includes table data in the backup or restore for matching table names only.

*Syntax*

```
-INCLUDE[_DATA] sql-regex
```

The backup or restore only includes data for table(s) matching the SQL regular expression argument. Opposite of `-SKIP_D[ATA]`.

The metadata of non-matching tables is included, only their data is skipped.

Selectively including data in a backup or restore can yield errors during restore when constraints are enabled, and a foreign key constraint exists on a table that was included, depending on a table that was not included.

You can use `-I[NACTIVE]` to disable **all** indices, primary key, unique key and foreign key constraints. If `CHECK` constraints depend on not included data, you may also need to specify `-N[O_VALIDITY]`.

The SQL regular expression syntax is documented in the *Firebird 5.0 Language Reference*.

Introduced in Firebird 5.0.

### 2.1.14. -ST[ATISTICS]

Show statistics.

*Syntax*

```
-ST[ATISTICS] options
```

Show statistics, with *options* one or more of:

`T` Time from start

`D` Delta time

`R` Page reads

`W` Page writes

For example, `-ST TDRW` will display all statistics.

The statistics are only displayed when `-V[ERIFY]` or `-VERBI[NT]` is specified.

Introduced in Firebird 2.5

### 2.1.15. -TR[USTED]

Use Windows trusted authentication (`Win_Sspi`).

Introduced in Firebird 3.0.

### 2.1.16. -USER

Username for authentication.

*Syntax*

```
-USER username
```

Allows the username of the SYSDBA or database owner user to be specified if the database is to be backed up, or, in the case of a restore (with the `-c[reate]` switch specified), any valid username can be specified. This need not be supplied if `ISC_USER` environment variable exists and has a correct value for the username.

Databases can only be backed up by SYSDBA, users with the `RDB$ADMIN_ROLE`, the database owner, or — since Firebird 4.0 — users with the `USE_GBAK_UTILITY` system privilege. A restore can also be carried out by SYSDBA or the database owner, however, if the `-c(reate)` switch is used, *any* authenticated user can create a new database from a backup file. In Firebird 3.0 and higher, non-admin users need the `CREATE DATABASE` DDL privilege to be able to restore a database.

### 2.1.17. -V[ERIFY]

Display information on the backup or restore.

Normally `gbak` operates quietly with no information written to the display. This switch reverses that situation and causes lots of information to be displayed. The default is to display the output to the screen, but you can redirect the output to a log file using the `-y` switch.

This option is mutually exclusive with `-VERBI[NT]`. Using `-verify` is the same as specifying `-verbint 10000`.

Contrary to its name, this option doesn't *verify* anything, and it would have been better named `-VERBOSE`.

The only way to verify a backup is to restore it, check it doesn't complete with errors, and maybe perform some queries for sanity checking

### 2.1.18. -VERBI[NT]

Verbose information with explicit interval.

*Syntax*

```
-VERBI[NT] interval
```

The *interval* controls at how many records written gbak will output information; in other words, it controls the frequency of the output of "... records written" messages. The minimum value is 100.

This option is mutually exclusive with -V[ERIFY]. Using -verify is the same as specifying -verbint 10000.

Introduced in Firebird 3.0.

### 2.1.19. -Y

Write -V[ERIFY] output to a log file.

*Syntax*

```
-Y { filename | SUPPRESS }
```

Used in conjunction with the -v[erify] switch to redirect status messages to a file or device, rather than the screen, or to suppress them altogether.

If -y suppress is used, then no information will be written to screen regardless of whether -v[erify] is specified.

If a filename is given *and* the -v[erify] switch is specified, the file will be written to record progress, errors etc.

All known documentation on this switch mentions that the text should be "suppress_output" rather than "suppress". This is an error in the documentation as the source code for gbak shows that the switch must be "suppress".

The log file must not exist. If it does, the backup or recovery operation will fail:

```
tux> rm employee.log
tux> gbak -backup employee.fdb employee.fbk -y employee.log -v

tux> ls -l employee.log
-rw-r--r-- 1 firebird firebird 21610 2010-08-04 10:22 employee.log

tux> gbak -backup employee.fdb employee.fbk -y employee.log -v
gbak:cannot open status and error output file employee.log
```

### 2.1.20. -Z

Display version information.

This option displays information about the version of gbak being used, and optionally, a database. To obtain the version of gbak only, run the command as follows:

```
tux> gbak -z

gbak:gbak version LI-V2.5.0.26074 Firebird 2.5
gbak: ERROR:requires both input and output filenames
gbak:Exiting before completion due to errors

tux> echo $?
1
```

This displays the current version of gbak, and after displaying a couple of error messages, exits with an error code of 1 as shown above by the echo command. This method does not attempt to backup any databases and does not require a username and password to be defined or supplied.

If you wish to display the version number of the gbak utility and also details of database, you must supply a valid database name *and* backup filename, as follows:

```
tux> gbak -z employee employee.fbk -user sysdba -password secret

gbak:gbak version LI-V2.1.3.18185 Firebird 2.1
gbak:    Version(s) for database employee
Firebird/linux (access method),version LI-V2.1.3.18185 Firebird 2.1
Firebird/linux (remote server),version LI-V2.1.3.18185
Firebird 2.1/tcp (tux)/P11
Firebird/linux (remote interface), version LI-V2.1.3.18185
Firebird 2.1/tcp (tux)/P11
on disk structure version 11.1

tux> echo $?
0
```

You will note from the above that a valid username and password must be defined on the command line, or by the use of environment variables ISC_USER and ISC_PASSWORD. This version of the command will exit with a error code of zero.

This method of calling gbak *will* make a backup of the database. If your database is large, this can take some time to complete and the backup file specified *will* be overwritten if it already exists. Beware.

The output above has been modified to allow it to fit the page width for a PDF.

## 2.2. Backup Switches

When running `gbak`, if the *first* filename is a database name, or database alias then `gbak` will default to taking a backup of the database in transportable format. The backup file will be named as per the second filename supplied on the command line.

You can also send the output to standard output rather than a backup file. In this case, you must specify `stdout` as the backup filename. This is not really of much use, unless you wish to pipe the backup through a tool to modify it in some way. You can pipe the output directly to a `gbak` restore operation to clone a database without needing an intermediate backup file. An example is given later in this manual.

When carrying out a backup of a database, the following switches, in addition to the common ones above, will be of use:

### 2.2.1. -B[ACKUP_DATABASE]

Backup a database.

### 2.2.2. -CO[NVERT]

Convert external tables to normal tables.

This switch causes any *external* tables to be backed up as if they were normal (non-external) tables. When this backup file is used to restore a database, the tables that were external in the original database will now be normal tables.

### 2.2.3. -E[XPAND]

Do not apply RLE compression on backup data.

Normally, `gbak` will compress the output file. This switch prevents that compression from taking place.

This is a very basic RLE compression with a low compression rate, for better compression, use `-ZIP`.

### 2.2.4. -FA[CTOR]

Blocking factor.

*Syntax*

```
-FA[CTOR] block-size
```

If backing up to a physical tape device, this switch lets you specify the tape's blocking factor.

### 2.2.5. -G[ARBAGE_COLLECT]

Disable garbage collection.

This switch prevents Firebird's garbage collection from taking place while `gbak` is running. Normally `gbak` connects to the database as any other connection would and garbage collection runs normally. Using this switch prevents garbage collection from running during the course of the backup. This can help speed up the backup.

### 2.2.6. -IG[NORE]

Ignore bad checksums.

This switch causes gbak to ignore bad checksums in the database. This can be used to attempt to backup a database that failed due to checksum errors. There is no guarantee that the data will be usable though, so it is best to take other precautions to preserve your data.

### 2.2.7. -L[IMBO]

Ignore limbo transactions.

If you have a two-phase transaction (e.g. across two different databases) that failed because a server died before the commit or rollback, but after the changes were prepared, you have a limbo transaction. This switch forces the backup to ignore data from such broken transactions. It should not be used for normal backups and only used, like the `-IG[NORE]` switch to attempt to recover from a failure.

### 2.2.8. -NT

Create non-transportable backup.

This switch turns off the `-T[RANSPORTABLE]` switch (which is on by default) and causes the backup file to be created using platform dependent formats. If you use this switch to create a backup then you can only restore the backup on a similar platform. You cannot, for example, take a backup file from Linux over to a Windows server, or from a little-endian system to a big-endian system.

### 2.2.9. -OL[D_DESCRIPTIONS]

Use old backup description format.

This switch is unlikely to be used. It has been deprecated. Its purpose is to force the backup to be made using the older InterBase versions' format of metadata descriptions.

### 2.2.10. -T[RANSPORTABLE]

Create transportable backup.

The default backup file format is transportable. Transportable backup files are written in a format known as *external data representation* (XDR) format. It is this format which allows a backup taken on a server of one type to be successfully restored on a server of another type.

Given this is the default, there is no real need to specify this option. You can use it for explicitness if you want.

### 2.2.11. -ZIP

Backup file is in zip (zlib) compressed format.

This is a backup-only switch; on restore, the compression is detected and decompressed automatically.

Introduced in Firebird 4.0.

## 2.3. Restore Switches

When running a gbak command, if the *first* filename is a database backup filename then gbak will default to running a recovery of the database provided that you specify one of -c[reate database], -rep[lace_database] or -r[ecreate_database] in order to make your intentions clear. The database will be restored to whatever filename is specified as the second filename on the command line.

You may read the backup data directly from standard input rather than a backup file. In this case, you must specify stdin as the backup filename. You could pipe a gbak backup operation directly to a gbak restore operation to clone a database without needing an intermediate backup file. An example is given later in this manual.

When carrying out a restore or replacement of a database, the following switches, in addition to the common ones above, will be of use:

### 2.3.1. -C[REATE_DATABASE]

Restore to a new database.

This switch creates a new database from the backup file. The database file must not exist or the restore will fail. Either this switch or -rep[lace_database] or -R[ECREATE_DATABASE] [O[VERWRITE]] must be specified.

### 2.3.2. -R[ECREATE_DATABASE] [O[VERWRITE]]

Restore to a new database, optionally allowing overwriting an existing database.

Create (or replace if o[verwrite] is used) the named database from the backup file. Unless the O[VERWRITE] option is specified, the database filename should not already exist or an error will occur.

This switch is deliberately abbreviated to -r to try to prevent unsuspecting DBAs from overwriting an existing database thinking that the -r was abbreviated from -restore. Now, it takes special effort

to manage this, especially as `-restore` was never actually a valid switch; in older versions `-r` was in fact an abbreviation of `-REP[LACE_DATABASE]` and it did this by *first* deleting the existing database and *then* recreating it from the backup.

Using `-r[ecreate_database] o[verwrite]` is effectively the same as using `-rep[lace_database]`.

Introduced in Firebird 2.0.

### 2.3.3. -REP[LACE_DATABASE]

Replace database from backup file, overwriting an existing database if it exists.

This switch used to be abbreviated to `-r` in Firebird 1.5 and older.

If the target database already exists, it will first be deleted (dropped), before the restore is performed.

The `-REP[LACE_DATABASE]` switch is deprecated and may be removed in a future Firebird version.

Use either `-C[REATE_DATABASE]` or `-R[ECREATE_DATABASE] [O[VERWRITE]]`. Specifically, use `-r o` or `-recreate_database overwrite` for the exact equivalent of this switch.

### 2.3.4. -BU[FFERS]

Configure the database page buffer (cache size).

*Syntax*

```
-BU[FFERS] number-of-pages
```

This switch sets the default database cache size (in number of database pages) for the database being restored. If a database is being overwritten then this setting will overwrite the previous setting for the cache size.

### 2.3.5. -FIX_FSS_D[ATA]

Fix malformed `UNICODE_FSS` data.

*Syntax*

```
-FIX_FSS_D[ATA] charset-name
```

This switch forces `gbak` to fix malformed `UNICODE_FSS` character data during a restore.

Malformed string data will be attempted to be read using the specified character set *charset-name*, and then transformed to `UNICODE_FSS`. Specifying the wrong character set name may result in logical corruption of your data.

Do not use this switch without a clear understanding of what it does. Incorrect use could corrupt your data instead of fixing things. Always keep a copy of the original database and its backup.

This switch, and the following one, should not be required under normal circumstances. However, if a restore operation fails with a "malformed string" error, the message output from `gbak` will refer the user to one or both of these switches to fix the malformed `UNICODE_FSS` data or metadata as appropriate.

Introduced in Firebird 2.5.

### 2.3.6. -FIX_FSS_M[ETADATA]

Fix malformed `UNICODE_FSS` metadata.

*Syntax*

```
-FIX_FSS_M[ETADATA] charset-name
```

This switch forces `gbak` to fix malformed `UNICODE_FSS` metadata during a restore.

Malformed metadata string will be attempted to be read using the specified character set *charset-name*, and then transformed to `UNICODE_FSS`. Specifying the wrong character set name may result in logical corruption of the strings in your metadata.

Do not use this switch without a clear understanding of what it does. Incorrect use could corrupt your database instead of fixing things. Always keep a copy of the original database and its backup.

This switch, and the preceding one, should not be required under normal circumstances. However, if a restore operation fails with a "malformed string" error, the message output from `gbak` will refer the user to one or both of these switches to fix the malformed `UNICODE_FSS` data or metadata as appropriate.

Introduced in Firebird 2.5.

### 2.3.7. -I[NACTIVE]

Do not activate indexes after restore.

This switch can be used to restore a database when a previous restore attempt failed due to index errors. All indexes in the restored database will be inactive, and as a consequence all primary key, unique key and foreign key constraints as well.

### 2.3.8. -K[ILL]

Kill (do not recreate) shadow database.

This switch restores the database but doesn't recreate any shadow files that existed previously.

### 2.3.9. -MO[DE]

Restore database in read-only or read/write mode.

*Syntax*

```
-MO[DE] { READ_ONLY | READ_WRITE }
```

This switch allows the database being restored to be set to the given access mode when opened. By default, the mode is taken from the database that was backed up.

This option should not be confused with the replica mode configured through `-REPLICA`. For example, a database created with `-REPLICA READ_ONLY` is still writable by the replicator connection, while a database created with `-MODE READ_ONLY` is not writable at all.

### 2.3.10. -N[O_VALIDITY]

Do not enable check constraints.

This switch is similar to the `-i[nactive]` switch above, except, it disables all *check* constraints in the restored database.

### 2.3.11. -NOD[BTRIGGERS]

Disable database triggers during restore.

Prevents the *database triggers* from firing on a restore. Database triggers are a feature of Firebird 2.0 and higher, and are different from *table triggers*.

Introduced in Firebird 2.1.

### 2.3.12. -O[NE_AT_A_TIME]

Restore table data with a transaction per table.

This switch restores data one table at a time, using a transaction per table. This can be useful when a previous restore failed due to data errors. Normally, a restore takes place in a single transaction with a single commit at the end of the restore. If the restore is interrupted for any reason, an empty database is the end result. Using the `-o[ne_at_a_time]` option uses a transaction for each table and commits after each table has been restored.

### 2.3.13. -P[AGE_SIZE]

Page size of the restored database.

*Syntax*

```
-P[AGE_SIZE] page-size
```

Use this switch to change the default database page size. By default, the database is restored using the same page size as the original database (as recorded in the backup file).

Depending on the version, valid page sizes are 1024, 2048, 4096, 8192, 16384 and 32768. Support for page sizes 1024 and 2048 was removed in Firebird 2.1. Support for page size 32768 was added in Firebird 4.0.

### 2.3.14. -REPLICA

Configures the replica mode of the restored database.

*Syntax*

```
-REPLICA { NONE | READ_ONLY | READ_WRITE }
```

The replica mode of a database is stored in the backup file. On restore, by default, this replica mode is set for the newly created database.

The `-REPLICA` switch explicitly sets the replica mode, overriding the value inherited from the backup. For example, `NONE` will make the database a primary (or normal) database, `READ_ONLY` marks the database as a read-only replica, and `READ_WRITE` a read/write replica.

After restore, the replica mode of a database can be changed with `gfix -replica { NONE | READ_ONLY | READ_WRITE } <database>`.

Replication itself is out of the scope of this manual.

This option should not be confused with the read-only or read/write mode configured through `-MO[DE]`, which governs whether a database is entirely read-only. For example, a database created with `-REPLICA READ_ONLY` is still writable by the replicator connection, while a database created with `-MODE READ_ONLY` is not writable at all.

Introduced in Firebird 4.0.

### 2.3.15. -USE_[ALL_SPACE]

Use all space in page.

This switch forces the restore to use 100% of each database page and thus not leave any room for changes. If you omit this switch, some space will be kept free for subsequent changes. Using this switch is likely to be only of practical use where the database is created and used in read-only mode and no updates to existing data are required.

Once a database has been restored with this option specified, *all* database pages will be filled to 100% and no free space will be left for updates. Using this option for a read/write database can cause performance problems due to record versions or record updates getting split over multiple pages.

Use of this switch sets a flag in the database header page to signal that *all* pages are to be filled to 100% — this also applies to any new pages created after the restore.

You can override this setting, using `gfix -use {full | reserve} database_name` where `full` uses 100% of each page and `reserve` reserves some space for subsequent updates.

See chapter *Database Page Space Utilization* in *Firebird Database Housekeeping Utility*. for more information.

# Chapter 3. Backup Mode

Before you consider using other tools to take a backup of your Firebird database, make sure that you know what the tools do and how a running database will be affected by them. For example, if you use *Winzip* to create a compressed copy of a database, and you do it when users are accessing the system, the chances of a successful restore of that database are slim. You must either always use the `gbak` or `nbackup` tools which know how the database works, enable the database backup mode with `ALTER DATABASE BEGIN BACKUP` (and end it with `ALTER DATABASE END BACKUP`), or, use `gfix` to shut the database down completely before you even attempt to backup the database file(s).

`Gbak` creates a consistent backup of the database by starting a transaction that spans the backup period. When the backup is complete, the transaction is ended and this means that the backup process can be run while users are working in the database. However, any transactions started after the backup process begins will not have any of their changed data written to the backup file. The backup will represent a copy of the entire database at the moment the backup began.

The backup file created by a default `gbak` backup is cross-platform (transportable), so a backup taken on a Windows server can be used to recreate the same database on a Linux server, or on any other platform supported by Firebird. This is not true of the copies of your database taken (while the database was closed or in backup-mode!) with tools such as *Winzip* etc. Those copies should only ever be used to restore a database on the same platform as the one copied.

Always backup the database with the version of `gbak` supplied with the running database server.

And one final thought on backups: regardless of the fact that the backup finished with no errors, exited with an error code of zero and all appears to be well, how do you actually know that the backup file created is usable? The short answer is, you don't. Whenever you have a valuable database — and they all should be — you are strongly advised to take your backup files and use them to create a test restore of a database either on the same server or even better, on a different one. Only by doing this can you be certain of a successful backup.

The following example shows a backup being taken on a server named *linux* and used to create a clone of the database on another Linux server named *tux* to make sure that all was well. First of all, the backup on *linux*:

```
linux> gbak -backup -verify -y backup.log employee employee.fbk
linux> gzip -9 employee.fbk
```

Note that the above `gbak` command can be written as follows, leaving out the `-b[ackup]` switch as `gbak` defaults to running a backup when no other suitable switches are specified:

```
linux> gbak -verify -y backup.log employee employee.fbk
```

Then, on the *tux* server:

```
tux> scp norman@linux:employee.fbk.gz ./

Using keyboard-interactive authentication.
Password:
employee.fbk.gz           |         19 kB |  19.3 kB/s | ETA: 00:00:00 | 100%

tux> gunzip employee.fbk.gz
tux> gbak -r o -verify -y restore.log employee.fbk employee.restore.test
```

At this point, the restore has worked and has overwritten the previous database known as `employee.restore.test`.

The actual location of the database for the database `employee.restore.test` is defined in the `aliases.conf` file in `/opt/firebird` on the server. In this test, it resolves to `/opt/firebird/databases/employee.restore.fdb`.

For further proof of reliability, the application may be tested against this clone of the live database to ensure all is well.

## 3.1. Speeding up the Backup

There are a couple of tricks you can use to speed up the backup. The first is to prevent the garbage collection from being carried out while the backup is running. Garbage collection clears out old record versions that are no longer required and this is usually covered by a sweep — manual or automatic — or by a full table scan of any affected table. As `gbak` accesses all rows in the tables being backed up, it too will trigger the garbage collection and, if there have been a lot of updates, can slow down the backup. To prevent garbage collection during the backup, use the `-g[arbage_collect]` option.

```
tux> gbak -backup -garbage_collect employee /backups/employee.fbk
```

The second option is to backup the database using the `-se[rvice]` option. Although this is used to perform remote backups, it can be used locally as well. Using this option can help speed up your backups. It simply avoids the data being copied over the TCP network which can slow down the actions of the backup.

```
tux> gbak -backup -service tux:service_mgr employee /backups/employee.fbk
```

The example above backs up the employee database, on the tux server, "remotely" using the service manager. The tux server is, of course, where the command is running, so it isn't really running remotely at all.

You can, of course, combine the `-g[arbage_collect]` and `-se[rvice]` options.

# Chapter 4. Restore Mode

Backups taken with the gbak application from one version of Firebird can be used by later versions of Firebird to restore the database, however, while this may result in an upgrade to the On-Disk Structure (ODS) for the database in question, the SQL Dialect will never be changed. If you backup a Firebird 1.0 dialect 1 database and then use the backup file to recreate it under Firebird 2.1, for example, the ODS will be updated to 11.1 but the SQL dialect will remain as 1.

Always restore the database with the version of gbak supplied with the database server you wish to run the (new) database under. However, gbak from Firebird 2.1 can be used to restore a database onto any version of Firebird.

You can, if you wish, change the SQL dialect using gfix.

Under normal circumstances, restoring a database takes place as a single transaction. If the restore is successful, a commit at the end makes the data permanent, if not, the database will be empty at the end.

The restore option -o[ne_at_a_time] will use a transaction for each table and if the table is restored with no errors, a commit is executed rendering that table permanent regardless of what happens with subsequent tables.

## 4.1. Restore Or Recreate?

Should a database be restored or replaced? Restoring a database is the process by which you take the existing file and delete prior to replacing it on disk with a backup copy. Gbak does this when you specify the -r[ecreate_database] o[verwrite] switch or the -rep[lace_database] switch. What is the difference?

If a database exists on disk, and you ask gbak to restore it using one of the two switches above, you might corrupt the database especially if the database is in use and has not been shut down using gfix. In addition, if you have only partially completed the restore of a database, and some users decide to see if they can log in, the database may well be corrupted.

Finally, if the restore process discovers that the backup file is corrupt, the restore will fail and your previously working database will be gone forever.

It can be seen that restoring a database can be a fraught experience.

For security, always recreate the database with a new name — a clone — and update the aliases.conf to reflect the new name. This way, your users will always refer to the database by the alias regardless of the actual filename on the server.

## 4.2. Malformed String Errors During Restores

During a restore operation, most likely when restoring a backup taken using an older gbak version, it is possible to see failure messages, in gbak's output, indicating malformed Unicode strings. The reason that these may occur is as explained by Helen Borrie:

> The source text of stored procedures (and several other types of object, such as CHECK constraints) is stored in a blob, as is the "compiled" BLR code. When you restore a database, the BLR is not recreated: the same BLR is used until next time you recreate or alter the object.
>
> Historically, the engine did not do the right thing regarding the transliteration of strings embedded in the source and the BLR. In v.2.1 and 2.5 a lot of work was done to address the international language issues, as you probably know. A side effect of this was that everything that was read from data and meta data became subject to "well-formedness" checks. Hence, on restoring, those previously stored source and BLR objects are throwing "malformed string" errors when gbak tries to read and write the data in these system table records. This very old bug affects user blobs, too, if they have been stored using character set NONE and the client is configured to read a specified character set to which the stored data could not be transliterated.
>
> In v.2.1 there were scripts in ../misc that you could run to repair the meta data blobs and also use as a template for repairing the similar errors in blobs in your user data. The repair switches were added to the gbak restore code in v.2.5 to do the same corrections to meta data and data, respectively, during the process of restoring a database for upgrade.

## 4.3. Speeding up the Restore

The restoration of a database, from a backup, can be made to execute quicker if the `-se[rvice]` option is used. Although this is used to perform remote restores, it can be used locally as well. It simply avoids the data being copied over the TCP network which can slow down the actions of the restore.

```
tux> gbak -r o -service tux:service_mgr /backups/employee.fbk employee
```

The example above backs up the employee database, on the tux server, "remotely" using the service manager. The tux server is, of course, where the command is running, so it isn't really running remotely at all.

# Chapter 5. Security Of Backups

As you have seen above, anyone, with a valid username and password, can restore a `gbak` database backup file provided that they are not overwriting an existing database (in Firebird 3.0 and higher, they will also need the `CREATE DATABASE` DDL privilege). This means that your precious data can be stolen and used by nefarious characters on their own servers, to create a copy of your database and see what your sales figures, for example, are like.

To try and prevent this from happening, you are advised to take precautions. You should also prevent backups from being accidentally overwritten before they have expired. Some precautions you can take are:

- Always set the backup file to be read-only after the backup is complete. This helps prevent the file from being overwritten.
- Alternatively, incorporate the date (and time) in your backup filenames.
- Keep backups in a safe location on the server. Storing backups in a location with restricted access helps reduce the chances of your backup files 'escaping' into the wild.
- Keep tape copies of your backups very secure. A locked safe or off-site location with good security is advisable. The off-site location will also be of use after a total disaster as the backups are stored in a separate location to the server they are required on.
- Backup to a partition or disk that has encryption enabled.
- Encrypt the backup file — supported by Firebird 4.0 and higher.
- Make sure that only authorised staff have access to areas where backups are kept.
- Always test your backups by cloning a database from a recent backup.

In Firebird 2.1, there is an additional security feature built into `gbak` and all the other command-line utilities. This new feature automatically hides the password if it is supplied on the command line using the `-password` switch. `Gbak` replaces the password with spaces — one for each character in the password. This prevents other users on the system, who could run the `ps` command and view your command line and parameters, from viewing any supplied password. In this manner, unauthorised users are unable to obtain the supplied password.

```
tux> gbak -b -user SYSDBA -passw secret employee /backups/employee.fbk
```

```
tux> ps efx| grep -i gba[k]
20724 ... gbak -backup -user SYSDBA -passw           employee employee.fbk
... (lots more data here)
```

You can see from the above that the password doesn't show up under Firebird 2.1 as each character is replaced by a single space. This does mean that it is possible for someone to work out how *long* the password *could* be and that might be enough of a clue to a dedicated cracker. Knowing the length of the required password does make things a little easier, so for best results use a random number of spaces between `-passw` and the actual password. The more difficult you make things for

the bad people on your network, the better.

# Chapter 6. Backup & Restore Recipes

The following recipes show examples of backup and restore tasks using `gbak`. These are probably the commonest cases that you will encounter as a DBA. All the examples use the `employee` database supplied with Firebird and the actual location is correctly configured in `aliases.conf`.

Each of the following recipes is run with the assumption that the environment variables `ISC_USER` and `ISC_PASSWORD` have been given suitable values. If you don't have these set, you will need to supply the appropriate options `-USER` and `-PAS[SWORD]`, or `-TR[USTED]` — Windows-only, on the commandline.

## 6.1. Backup & Restore Prerequisites

If you replace an open and running database, there is a good chance that you will corrupt it. For best results and minimal chance of corrupting a database, you should close it before replacing it. To close a database, use `gfix` as follows:

```
tux> gfix -shut -tran 60 employee
```

The example above prevents any new transaction from being started which prevents new queries being executed or new sessions connecting to the database. It will wait for up to 60 seconds for everyone to logout and for all current transactions to complete before shutting down the database. If any long-running transactions have not completed by the end of 60 seconds, the shutdown will timeout and the database will remain open.

After the restore of the database has completed, the database will automatically be opened again for use.

## 6.2. A Simple Backup & Restore

This example takes a backup, then immediately overwrites the original database using the new backup. This is not normally a good idea as the first action of the `-recreate overwrite` is to wipe out the database.

```
tux> # Backup the database.
tux> gbak -backup employee /backups/employee.fbk

tux> # Restore the database.
tux> gfix -shut -tran 60 employee
tux> gbak -recreate overwrite /backups/employee.fbk employee
```

## 6.3. Metadata Only

It is possible to use `gbak` to recreate an empty database containing only the various *domains*, *tables*,

*indices* and so on, of the original database but none of the data. This can be useful when you have finished testing your application in a test environment and wish to migrate the system to a production environment, for example, but starting afresh with none of your test data.

```
tux> #Backup only the database metadata.
tux> gfix -shut -tran 60 employee
tux> gbak -backup -meta_data employee employee.meta.fbk
```

When the above backup file is restored on the production server, only the metadata will be present.

There is another way to create a database with no data and only the metadata. Simply restore from an existing backup which contains the data and supply the `-M[ETA_DATA]` switch to the restore command line. The database will be restored but none of the original data will be present.

```
tux> #Restore only the database metadata.
tux> gbak -create employee.fbk mytest.fdb -meta_data
```

The `-M[ETA_DATA]` switch can be used on either a backup or a restore to facilitate the creation of a clone database (or overwrite an existing one) with no actual data.

## 6.4. Splitting The Backup

The `gsplit` filter application, documented in its own manual, doesn't actually work anymore. This filter was supplied with old versions of InterBase and Firebird to allow large database backups to be split over a number of files so that file system limits could be met. Such limits could be the size of a CD, the 2GB limit on individual file sizes on a DVD, where some Unix file systems have a 2 GB limit and so on.

`Gbak` allows the backup files to be split into various sizes (with a minimum of 2048 bytes) and will only create files it needs.

```
tux> # Backup the database to multiple files.
tux> gbak -backup employee /backups/emp.a.fbk 600m /backups/emp.b.fbk 600m
```

The sizes after each filename indicate how large that particular file is allowed to be. The default size is bytes, but you can specify a suffix of `k`, `m` or `g` to use units of kilo, mega or gigabytes.

If the backup completes before writing to some files, then those files are not created. A backup file is only ever created when it must be.

The size of the final backup file will be quietly ignored if the database has grown too large to allow a truncated backup to complete. If, in the example above, the backup needs a total of 1500M, then the last file would be written to a final size of 900m rather than the 600m specified.

To restore such a multi-file backup requires that you specify all filenames in the backup and in *the correct order*. The following example shows the employee database above being restored from the

two files backed up above:

```
tux> # Restore the database from multiple files.
tux> gfix -shut -tran 60 employee
tux> gbak -r o /backups/employee.a.fbk /backups/employee.b.fbk employee
```

## 6.5. Changing The ODS

Normally the ODS used is the one in force by the version of Firebird used to restore the database. So, the examples above will actually change the ODS when the database is restored. The backup should be taken using the `gbak` utility supplied by the old ODS version of Firebird. The restore should be carried out using `gbak` from the newer version of Firebird.

```
tux> setenv_firebird 2.0
Firebird environment set for version 2.0.

tux> # Check current ODS version (as root user!)
tux> gstat -h employee|grep ODS
        ODS version             11.0

tux> # Backup the (old) database.
tux> gbak -backup employee /backups/employee.2_0.fbk

tux> setenv_firebird 2.1
Firebird environment set for version 2.1.

tux> # Recreate the database and upgrade the ODS.
tux> gfix -shut -tran 60 employee
tux> gbak -r o /backups/employee.2_0.fbk employee

tux> # Check new ODS version (as root user!)
tux> gstat -h employee|grep ODS
        ODS version             11.1
```

After the above, the old 2.0 Firebird database will have been recreated — wiping out the old database — as a Firebird 2.1 database with the corresponding upgrade to the ODS from 11.0 to 11.1.

The script `setenv_firebird` is not supplied with Firebird and simply sets `PATH`, etc., to use the correct version of Firebird as per the supplied parameter.

## 6.6. Changing The Cache Size

The default database cache (page buffer) is set when the database is created, or subsequently by using `gfix -b[uffers] <number-of-pages>`. `Gbak` can restore a database and set the default cache size as well. The process is as follows:

```
tux> # Check current cache size (as root user!)
tux> gstat -h employee | grep -i buffer
        Page buffers            0

tux> # Restore the database & change the cache size.
tux> gfix -shut -tran 60 employee
tux> gbak -r o -buffer 200 /backups/employee.fbk employee

tux> # Check the new cache size (as root user!)
tux> gstat -h employee | grep -i buffer
        Page buffers            200
```

The default cache size is used when the number of buffers is zero, as in the first example above. `Gbak` allows this to be changed if desired. `Gbak`, however, cannot set the cache size back to zero. You must use `gfix` to do this.

## 6.7. Changing The Page Size

Similar to the example above to change the default database cache size, the database page size can also be changed using `gbak`.

```
tux> # Check current page size (as root user!)
tux> gstat -h employee | grep -i "page size"
        Page size               4096

tux> # Restore the database & change the page size.
tux> gfix -shut -tran 60 employee
tux> gbak -r o -page_size 8192 /backups/employee.fbk employee

tux> # Check the new page size (as root user!)
tux> gstat -h employee | grep -i "page size"
        Page size               8192
```

## 6.8. Creating A Read-Only Database Clone

Sometimes you do not want your reporting staff running intensive queries against your production database. To this end, you can quite easily create a clone of your production database on a daily basis, and make it read-only. This allows the reporting team to run as many intensive reports as they wish with no ill effects on the production database, and it prevents them from inadvertently making changes.

The following example shows the production employee database running on Linux server *tux*, being cloned to the reporting team's Linux server named *tuxrep*. First on the production *tux* server:

```
tux> # Backup the production database.
```

```
tux> gbak -backup employee /backups/employee.fbk
```

Then on the reporting team's *tuxrep* server:

```
tuxrep> # Scp the backup file from tux.
tuxrep> scp fbuser@tux:/backups/employee.fbk ./
Using keyboard-interactive authentication.
Password:
employee.fbk              |         19 kB |  19.3 kB/s | ETA: 00:00:00 | 100%

tuxrep> # Restore the employee database as read-only.
tuxrep> gfix -shut -tran 60 employee
tuxrep> gbak -r o -mode read_only employee.fbk employee

tuxrep> # Check database mode (as root user)
tuxrep> gstat -h employee|grep -i attributes
        Attributes              no reserve, read only
```

## 6.9. Creating a Database Clone Without a Backup File.

You may use `gbak` to create a clone of a database, on the same server, without needing to create a potentially large backup file. To do this, you pipe the output of a `gbak` backup directly to the input of a `gbak` restore, as follows.

```
tux> # Clone a test database to the same server, without requiring a backup file.
tux> gbak -backup emptest stdout | gbak -r o stdin emptest_2
```

You will notice that the output filename for the backup is `stdout` and the input filename for the restore is `stdin`. This ability to pipe standard output of one process to the standard input of another, is how you can avoid creating an intermediate backup file. The commands above assume that there are suitable alias names set up for both emptest and emptest_2. If not, you will need to supply the full path to the two databases rather than the alias.

The `-r o` option on the restore process will overwrite the database name specified — as an alias or as a full path — if it exists and will create it anew if it doesn't.

If you don't want to overwrite any existing databases, use `-C[REATE_DATABASE]` which will only create a database if it doesn't already exist, and will exit with an error if it does. In POSIX compatible systems, the error code in `$?` is 1 in this case.

Further examples of backing up and restoring remote databases over ssh, using the `stdin` and `stdout` filenames, can be seen below.

## 6.10. Backup & Restore With & Without Shadow Files.

Databases can have shadow files attached in normal use. `Gbak` happily backs up and restores those

as well and in normal use, shadow files will be recreated. Should you wish to restore the database only and ignore the shadows, gbak can do that for you as the following example shows.

```
tux> # Check current shadows, use isql as gstat is broken.
tux> isql employee

Database:  employee
SQL> show database;
Database: employee
        Owner: SYSDBA
 Shadow 1: "/opt/firebird/shadows/employee.shd1" manual
 Shadow 2: "/opt/firebird/shadows/employee.shd2" manual
...

SQL> quit;

tux> # Restore the database preserving shadow files.
tux> gfix -shut -tran 60 employee
tux> gbak -recreate overwrite /backups/employee.fbk employee

tux> # Check shadows again, use isql as gstat is broken.
tux> isql employee

Database:  employee
SQL> show database;
Database: employee
        Owner: SYSDBA
 Shadow 1: "/opt/firebird/shadows/employee.shd1" manual
 Shadow 2: "/opt/firebird/shadows/employee.shd2" manual
...

SQL> quit;


tux> # Restore the database killing shadow files.
tux> gfix -shut -tran 60 employee
tux> gbak -recreate overwrite -kill /backups/employee.fbk employee

tux> # Check shadows again, use isql as gstat is broken.
tux> isql employee

Database:  employee
SQL> show database;
Database: employee
        Owner: SYSDBA
...

SQL> quit;
```

I use `isql` in the above examples as `gstat -h` seems to get confused about how many shadows there are on a database. It reports zero when there are two, eventually it catches up and reports that there are two, then, if you kill a shadow, it reports that there are now three!

## 6.11. Remote Backups & Restores

Firebird's `gbak` utility can make backups of a remote database. To do this, you need to connect to the service manager running on the remote server, this is normally called `service_mgr`. The following example shows the Firebird `employee` database on server *tuxrep* being backed up from the server *tux*. The backup will be written to the remote server, in other words, the backup file will be created on the *tuxrep* server and not on the *tux* one. The network protocol in use is TCP.

```
tux> # Backup the reporting database on remote server tuxrep.
tux> gbak -backup -service tuxrep:service_mgr employee /backups/remote_backup.fbk
```

The backup file will have the same owner and group as the Firebird database server — on Unix systems at least.

It is also possible to restore a remote database in this manner as well, and `gbak` allows this.

```
tux> # Restore the read-only reporting database on remote server tuxrep.
tux> gbak -r o -mode read_only -service tuxrep:service_mgr \
           /backups/remote_backup.fbk employee
```

The above example uses the handy Unix ability to split a long line over many shorter ones using a backslash as the *final* character on the line.

As ever, you are advised to beware of replacing a database in case there are problems during the restore. The example above recreates the existing database in read-only mode but this need not always be the case.

A remote backup can also be run on the database server itself! On Windows, this makes no difference, but on Unix systems, this local-remote method of backups and restores reduces network traffic. The 'remote' server, in this case, is not actually remote it is just the method of running the backup — connecting to the service manager — that implies remoteness.

```
tux> # Backup the employee database on this server, but pseudo-remotely!
tux> gbak -backup -service tux:service_mgr employee /backups/remote_backup.fbk
```

And corresponding restores can also be run 'remotely':

```
tux> # Restore the employee database on this server, but pseudo-remotely!
tux> gbak -r o -service tux:service_mgr /backups/remote_backup.fbk employee
```

The format of the parameter used for the `-service` switch is different according to the nature of the network protocol and the connection string in use:

**TCP**

Legacy connection string: `<hostname>[/<port>]:service_mgr`
Firebird 3.0 and higher connection strings:

```
# TCP/IP (v4 and v6)
INET://[<hostname>[:<port]]/service_mgr
# TCP/IP (v4 only)
INET4://[<hostname>[:<port]]/service_mgr
# TCP/IP (v6 only)
INET6://[<hostname>[:<port]]/service_mgr
```

If port is not specified, port `3050` is used. If hostname is not specified, `localhost` is used. It is not possible to specify a port without hostname.

To use a IPv6 IP address it must be enclosed in square brackets (e.g. `inet://[::1]/service_mgr`).

**XNET**

Legacy connection string: `service_mgr` (this may also use TCP/IP to localhost instead)
Firebird 3.0 and higher connection string: `xnet://service_mgr`

XNET is only supported on Windows.

**Named pipes/WNET**

Legacy connection string: `\\<hostname>[@<port>]\service_mgr`
Firebird 3.0 and 4.0 connection string: `WNET://[<hostname>[:<port>]]/service_mgr`

Support for named pipes (a.k.a. WNET) was removed in Firebird 5.0, and in earlier versions only available on Windows.

Since Firebird 4.0, the `service_mgr` name is no longer required and may now be left out of the connection string.

That means that the minimum connection strings to use the service manager are now:

- `.` or effectively anything that is not interpreted as one of the other connection strings (using XNET or TCP/IP)
- `xnet://`
- `inet://`, `inet4://`, `inet6://` (using localhost)
- `wnet://` (using localhost)

Be aware that not specifying a value for `-SE[RVICE]` is not possible. If the commandline has any other options or values after `-se`, this will be silently ignored and instead consume the next commandline option as its value, effectively

ignoring that value.

## 6.12. Remote Backups and Restores Using SSH

As shown above, you can use the special filenames `stdin` and `stdout` to backup and restore a database to a separate database on the same server. However, you can also use the same tools, over an SSH connection to a remote server, and pass the backup of one database directly to a restoration of a separate one.

The first example copies a local database to a remote server where Firebird is running and the firebird user has its environment set up so that the gbak tool is on `$PATH` by default, on login.

In each of the following examples, the `-user sysdba` and `-password whatever` parameters on the command lines have been replaced by `{...}`. When executing these commands, any remote `gbak` commands will require to have them specified unless the firebird user on the remote database(s) has `ISC_USER` and `ISC_PASSWORD` defined in the `.profile` or `.bashrc` (or equivalent) login files. However, that is a *seriously* bad idea and incredibly insecure.

```
tux> # Clone a test database to a different server, without requiring a backup file.
tux> gbak -backup employee stdout | \
ssh firebird@tuxrep "gbak {...} -r o stdin emptest"
```

When the above is executed, you will be prompted for a password for the remote firebird user on server *tuxrep*, assuming that you don't have a proper SSH key-pair already set up and active. The command will replace the local database according to the alias name `emptest`, but you can, if required, supply full path names for the databases. The following shows an example of the above being executed.

```
tux> # Clone a test database to a different server, without requiring a backup file.
tux> gbak -backup employee stdout | \
ssh firebird@tuxrep "gbak {...} -r o stdin emptest"

firebird@tuxrep's password:
```

As you can see, there's not much in the way of output, but you can connect remotely and check:

```
tux> isql {...} tuxrep:emptest

Database:  tuxrep:emptest

SQL> show database;

Database: tuxrep:emptest
        Owner: SYSDBA
PAGE_SIZE 4096
```

```
...
```

The next example, shows a remote database being backed up to a local one, in a similar manner.

```
tux> ssh firebird@tuxrep "gbak -backup {...} emptest stdout" | \
gbak -create stdin data/tuxrep_emptest.fdb

firebird@tuxrep's password:

tux> ls data

employee.fdb  tuxrep_emptest.fdb
```

You can see that a new `tuxrep_emptest.fdb` database has been created. Does it work? Checking with `isql` shows that it does.

```
tux> isql data/tuxrep_emptest.fdb

Database:  data/tuxrep_emptest.fdb

SQL> quit;
```

The final example shows how to backup a remote database on one server, to a remote database on another.

```
tux> ssh firebird@tuxrep "gbak -backup {...} emptest stdout" |  \
ssh firebird@tuxqa "gbak -create {...} stdin data/tuxrep_empqa.fdb"

firebird@tuxrep's password:
firebird@tuxqa's password

tux> ssh firebird@tuxqa "ls data"

employee.fdb  tuxrep_empqa.fdb
```

## 6.13. Using External Tools

`Gbak` and `nbackup` are the best tools to use when backing up and/or restoring Firebird databases. They have been extensively tested and know the internals of the database and how it works, so the chances of these tools corrupting your valuable data are very slim. However, some DBAs still like to use external tools (those not supplied with Firebird) to make backups for whatever reason.

Because it is not possible for external tools to know where a database is to be found, given the alias name, the scriptwriter and/or DBA must explicitly find out the correct location of the database file(s) and supply these to the external tool. To make this simpler for scriptwriters, my own installation uses a standard in my `aliases.conf` file as follows:

- The database alias must start in column one.
- There must be a single space before the equals sign (=).
- There must be a single space after the equals sign (=).
- Double quotes around the database filename is not permitted — it doesn't work for the Firebird utilities either.
- Databases are all single file databases.

The last rule applies to my installation only and means that the following simple backup script will work. If multiple file databases were used, more coding would be required to take a backup using external tools.

```
tux> cat /opt/firebird/aliases.conf
# ----------------------------------------------------------
# WARNING: Backup Standards require that:
#          The database name starts in column 1.
#          There is a single space before the equals sign.
#          There is a single space after the equals sign.
#          The path has no double quotes (they don't work!)
# ----------------------------------------------------------
employee = /opt/firebird/examples/empbuild/employee.fdb
```

The following shows the use of the `gzip` utility on a Linux server to take and compress a backup of a running database. The following is run as the root user due to the requirement to run `gfix` to shut down the database.

```
tux> # Backup the production employee database using gzip.
tux> gfix -shut -tran 60 employee
tux> DBFILE=`grep -i "^employee =" /opt/firebird/aliases.conf | cut -d" " -f3`
tux> gzip -9 --stdout $DBFILE > /backups/employee.fdb.gz
```

The restore process for this database would be the reverse of the above. Again, the following runs as root.

```
tux> # Restore the production employee database from a gzip backup.
tux> gfix -shut -tran 60 employee
tux> DBFILE=`grep -i "^employee =" /opt/firebird/aliases.conf | cut -d" " -f3`
tux> gunzip --stdout /backups/employee.fdb.gz > $DBFILE

tux> # Make sure firebird can see the file.
tux> chown firebird:firebird $DBFILE
```

# Chapter 7. Gbak Caveats

The following is a brief list of gotchas and funnies that I have detected in my own use of gbak. Some of these are mentioned above, others may not be. By collecting them all here in one place, you should be able to find out what's happening if you have problems.

## 7.1. Gbak Default Mode

If you do not specify a mode switch such as -b[ackup] or -c[reate] etc, then gbak will perform a backup as if the -b[ackup] switch had been specified — provided that the other switches specified are correct for a backup.

This detection of whether you are attempting a backup or a restore means that if you use the -z command line switch to view gbak information, then you *will* create a backup — and in Firebird 1.5 and older, overwrite the backup file you supply — if the command line also has a database name and a backup filename present. This assumes that there is a way for gbak to determine the username and password to be used — either as command line parameters or via defined environment variables.

## 7.2. Normal Versus Privileged Users

Only a SYSDBA, a user with the RDB$ADMIN role, the owner of a database, or a user with the USE_GBAK_UTILITY system privilege can take a backup of the database. However, *any* authenticated user can restore a database backup using the -c[reate] switch (in Firebird 3.0 and higher, this user will need the CREATE DATABASE DDL privilege). This means that you must make sure you prevent your backup files from falling into the wrong hands because there is nothing then to stop unauthorised people from seeing your data by the simple process of restoring *your* backups onto *their* server.

The database restore will fail, of course, if the user carrying it out is not the database owner and a database with the same filename already exists.

## 7.3. Silent Running?

The -y suppress_output switch is supposed to cause all output to be suppressed. Similar in fact to running with -v[erify] not specified. However, all it seems to do is cause the output (according to the -v[erify] switch setting) to be written to a file called suppress_output, however this only works once because the next run of gbak with -y suppress_output will fail because the file, suppress_output, already exists.

It is possible that this problem was introduced at version 2 for Firebird, because both 2.0 and 2.1 versions actually use the -y suppress switch rather then -y suppress_output. Using this (shorter) option does work as intended and the output is indeed suppressed.

## 7.4. Gbak Log File Cannot Be Overwritten

If you specify a log filename with the `-y <log file>` switch, and the file already exists, then even though the firebird user owns the file, and has write permissions to it, `gbak` cannot overwrite it. You must always specify the name of a log file that doesn't exist. On Linux systems, the following might help:

```
tux> # Generate unique backup and log filename.
tux> FILENAME=employee_`date "+%Y%m%d_%H%M%S"`

tux> # Shut down and Backup the database
tux> gfix -shut -tran 60 employee
tux> gbak -backup employee /backups/${FILENAME}.fbk -y /logs/${FILENAME}.log -v
```

The above is quite useful in as much as it prevents you from overwriting previous backups that may be required. The downside is that you now need to introduce a housekeeping system to tidy away old, unwanted backups to prevent your backup area filling up.

## 7.5. Use of `stdin` or `stdout` Filenames

`Gbak` recognizes the literal strings `stdin` and `stdout` as source or destination filenames. In POSIX systems, when the standard input and/or standard output channels are used, it is not permitted to execute seek operations on these channels. Using `stdin` or `stdout` as filenames with `gbak` will force `gbak` to use processing that will not seek on the input or output channels, making them suitable for use in pipes — as per the examples in the recipes section above.

These filenames, while they appear to be POSIX names, are definitely not synonyms for `/dev/stdin` or `/dev/stdout`, they are simply literals that `gbak` checks for while processing its parameters. Do not attempt to use names `/dev/stdin` or `/dev/stdout` in a piped process as it will most likely fail.

If you wish to create a backup file actually named either `stdin` or `stdout`, then you should specify the filename as a full, or relative, path name such as `./stdin` or `./stdout`, which causes `gbak` to treat them as a literal filename rather than a special filename that causes different to normal processing during the backup or restore process.

# Appendix A: Document history

The exact file history is recorded in the firebird-documentation git repository; see https://github.com/FirebirdSQL/firebird-documentation

**Revision History**

| | | | |
|---|---|---|---|
| 1.16 | 24 February 2024 | MR | Fixed rendering issue with `-INCLUDE[_DATA]` syntax |
| 1.15 | 24 February 2024 | MR | Include `gbak` name in the document title, making it the same as on https://firebirdsql.org/en/reference-manuals/ |
| 1.14 | 17 Feb 2024 | MR | • Fix wrong rendering due to wrong double quotes<br>• `-STATISTICS` also displayed for `-VERBINT`<br>• `-USER` cannot be abbreviated to `-U`<br>• Added notes on `-REPLICA` vs `-MODE`<br>• Added word-joiner in commandline switches between *minus* (-) and first character to ensure they aren't broken up on word wrap<br>• Add links from usage instruction to relevant sections |
| 1.13 | 16 Feb 2024 | MR | • Reordered document history so most recent changes are on the top<br>• Some copy-editing and formatting of the document history<br>• Added common options added in Firebird 3.0, 4.0, and 5.0, and some from 2.5 that weren't documented yet (documented with minimal information, will be expanded later)<br>• Removed common options repeated with "see above" in the backup and restore options<br>• Reordered restore options to put the main restore options (`-CREATE_DATABASE` and `-RECREATE_DATABASE [OVERWRITE]`) first<br>• Misc. copy-editing of commandline options<br>• Convert commandline options from definition lists to sections<br>• Replaced usage of `-REPLACE_DATABASE` (or its abbreviations) in examples with `-RECREATE_DATABASE OVERWRITE` (or abbreviations)<br>• Added more information on `-CRYPT`, `-DIRECT_IO`, `-KEYHOLDER`, `-KEYNAME`, `-PARALLEL`, `-REPLICA`, `-VERBINT` |
| 1.12 | 18 Jun 2020 | MR | Conversion to AsciiDoc, minor copy-editing |
| 1.11 | 1 May 2013 | ND | A correction to the above change to the `-use_[all_space]` command line switch — it affects all subsequent pages as well as the ones created during the restore. |
| 1.10 | 1 May 2013 | ND | Slight update to the `-use_[all_space]` command line switch, to explain how it works in a more understandable manner. |

**Revision History**

| | | | |
|---|---|---|---|
| 1.9 | 11 Apr 2013 | ND | • A section has been added to explain how to speed up your backups.<br>• A note has been added to the `-service` option to explain that it's use is not restricted to remote databases.<br>• Syntax errors in some examples corrected. |
| 1.8 | 14 Jan 2013 | ND | Further updates to document the use of the stdin and stdout filenames in backups and restores. A section has been added to Gbak Caveats giving more in depth detail about these two special filenames. |
| 1.7 | 11 Jan 2013 | ND | • Updated to document the use of the stdin and stdout filenames in backups and restores, which allow backups to be written to or read from standard input and standard output.<br>• A section was added on the use of the above to clone databases without requiring an intermediate backup file.<br>• An additional section was also added to show how — using the above in conjunction with SSH — backup and/or restore operations could be carried out on databases where one or both of the databases in question, are remote.<br>• A few minor formatting errors, URLs and some examples were corrected.<br>• Also added an example of a metadata only backup and restore. |
| 1.6 | 11 Oct 2011 | ND | *Updated to cover Firebird 2.5 changes. * Corrected description of `-g[arbage_collect]` switch. * Lots of spelling mistakes corrected. |
| 1.5 | 31 Mar 2011 | ND | Updated the `-z` option to indicate that it *does* carry out a backup. |
| 1.4 | 09 Aug 2010 | ND | Noted that `gbak` defaults to running a backup or recover according to the first filename parameter supplied. |
| 1.3 | 24 Jun 2010 | ND | Added a bit more details to the `-o[ne_at_a_time]` restore option to explain transactions. |
| 1.2 | 24 Nov 2009 | ND | Corrected the section on `-y Suppress_output` plus corrected the formatting of various screen dumps. They had been reformatted as text at some point. |
| 1.1 | 20 Oct 2009 | ND | More minor updates and converted to a standalone manual. |
| 1.0 | 10 Oct 2009 | ND | Created as a chapter in the Command Line Utilities manual. |

# Appendix B: License notice



The Original Documentation is titled *Firebird Backup & Restore Utility*.

The Initial Writer of the Original Documentation is: Norman Dunbar.



Contributor(s): Mark Rotteveel.