```
shell> mysqlbinlog copied-binlog.000001 copied-binlog.000002 | mysql -u roo
```

For more information, see Section 4.6.9.3, “Using mysqlbinlog to Back Up Binary Log Files”.

This method has the advantage that a new server is available almost immediately; only those transactions that were committed while the snapshot or dump file was being replayed still need to be obtained from the existing source. This means that the replica's availability is not instantaneous, but only a relatively short amount of time should be required for the replica to catch up with these few remaining transactions.

Copying over binary logs to the target server in advance is usually faster than reading the entire transaction execution history from the source in real time. However, it may not always be feasible to move these files to the target when required, due to size or other considerations. The two remaining methods for provisioning a new replica discussed in this section use other means to transfer information about transactions to the new replica.

Injecting empty transactions. The source's global `gtid_executed` variable contains the set of all transactions executed on the source. Rather than copy the binary logs when taking a snapshot to provision a new server, you can instead note the content of `gtid_executed` on the server from which the snapshot was taken. Before adding the new server to the replication chain, simply commit an empty transaction on the new server for each transaction identifier contained in the source's `gtid_executed`, like this:

```
SET GTID_NEXT='aaa-bbb-ccc-ddd:N';

BEGIN;
COMMIT;

SET GTID_NEXT='AUTOMATIC';
```

Once all transaction identifiers have been reinstated in this way using empty transactions, you must flush and purge the replica's binary logs, as shown here, where *`N`* is the nonzero suffix of the current binary log file name:

```
FLUSH LOGS;
PURGE BINARY LOGS TO 'source-bin.00000N';
```

You should do this to prevent this server from flooding the replication stream with false transactions in the event that it is later promoted to the source. (The `FLUSH LOGS` statement forces the creation of a new binary log file; `PURGE BINARY LOGS` purges the empty transactions, but retains their identifiers.)

This method creates a server that is essentially a snapshot, but in time is able to become a source as its binary log history converges with that of the replication stream (that is, as it catches up with the source or sources). This outcome is similar in effect to that obtained using the remaining provisioning method, which we discuss in the next few paragraphs.

Excluding transactions with gtid_purged. The source's global `gtid_purged` variable contains the set of all transactions that have been purged from the source's binary log. As with the method discussed previously (see Injecting empty transactions), you can record the value of `gtid_executed` on the server from which the snapshot was taken (in place of copying the binary logs to the new server). Unlike the previous method, there is no need to commit empty transactions (or to issue `PURGE BINARY LOGS`); instead, you can set `gtid_purged` on the replica directly, based on the value of `gtid_executed` on the server from which the backup or snapshot was taken.

As with the method using empty transactions, this method creates a server that is functionally a snapshot, but in time is able to become a source as its binary log history converges with that of the source and other replicas.