# 7.12 Git Tools - Bundling

## Bundling

Though we've covered the common ways to transfer Git data over a network (HTTP, SSH, etc), there is actually one more way to do so that is not commonly used but can actually be quite useful.

Git is capable of "bundling" its data into a single file. This can be useful in various scenarios. Maybe your network is down and you want to send changes to your co-workers. Perhaps you're working somewhere offsite and don't have access to the local network for security reasons. Maybe your wireless/ethernet card just broke. Maybe you don't have access to a shared server for the moment, you want to email someone updates and you don't want to transfer 40 commits via `format-patch`.

This is where the `git bundle` command can be helpful. The `bundle` command will package up everything that would normally be pushed over the wire with a `git push` command into a binary file that you can email to someone or put on a flash drive, then unbundle into another repository.

Let's see a simple example. Let's say you have a repository with two commits:

```
$ git log
commit 9a466c572fe88b195efd356c3f2bbeccdb504102
Author: Scott Chacon <schacon@gmail.com>
Date:   Wed Mar 10 07:34:10 2010 -0800

    Second commit

commit b1ec3248f39900d2a406049d762aa68e9641be25
Author: Scott Chacon <schacon@gmail.com>
Date:   Wed Mar 10 07:34:01 2010 -0800

    First commit
```

If you want to send that repository to someone and you don't have access to a repository to push to, or simply don't want to set one up, you can bundle it with `git bundle create`.

```
$ git bundle create repo.bundle HEAD master
Counting objects: 6, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 441 bytes, done.
Total 6 (delta 0), reused 0 (delta 0)
```

Now you have a file named `repo.bundle` that has all the data needed to re-create the repository's `master`branch. With the `bundle` command you need to list out every reference or specific range of commits that you want to be included. If you intend for this to be cloned somewhere else, you should add HEAD as a reference as well as we've done here.

You can email this `repo.bundle` file to someone else, or put it on a USB drive and walk it over.

On the other side, say you are sent this `repo.bundle` file and want to work on the project. You can clone from the binary file into a directory, much like you would from a URL.

```
$ git clone repo.bundle repo
Cloning into 'repo'...
...
$ cd repo
$ git log --oneline
9a466c5 Second commit
b1ec324 First commit
```

If you don't include HEAD in the references, you have to also specify `-b master` or whatever branch is included because otherwise it won't know what branch to check out.

Now let's say you do three commits on it and want to send the new commits back via a bundle on a USB stick or email.

```
$ git log --oneline
71b84da Last commit - second repo
c99cf5b Fourth commit - second repo
7011d3d Third commit - second repo
9a466c5 Second commit
b1ec324 First commit
```

First we need to determine the range of commits we want to include in the bundle. Unlike the network protocols which figure out the minimum set of data to transfer over the network for us, we'll have to figure this out manually. Now, you could just do the same thing and bundle the entire repository, which will work, but it's better to just bundle up the difference - just the three commits we just made locally.

In order to do that, you'll have to calculate the difference. As we described in [Commit Ranges](#), you can specify a range of commits in a number of ways. To get the three commits that we have in our `master` branch that weren't in the branch we originally cloned, we can use something like `origin/master..master` or `master ^origin/master`. You can test that with the `log` command.

```
$ git log --oneline master ^origin/master
71b84da Last commit - second repo
c99cf5b Fourth commit - second repo
7011d3d Third commit - second repo
```

So now that we have the list of commits we want to include in the bundle, let's bundle them up. We do that with the `git bundle create` command, giving it a filename we want our bundle to be and the range of commits we want to go into it.

```
$ git bundle create commits.bundle master ^9a466c5
Counting objects: 11, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (9/9), 775 bytes, done.
Total 9 (delta 0), reused 0 (delta 0)
```

Now we have a `commits.bundle` file in our directory. If we take that and send it to our partner, she can then import it into the original repository, even if more work has been done there in the meantime.

When she gets the bundle, she can inspect it to see what it contains before she imports it into her repository. The first command is the `bundle verify` command that will make sure the file is actually a valid Git bundle and that you have all the necessary ancestors to reconstitute it properly.

```
$ git bundle verify ../commits.bundle
The bundle contains 1 ref
71b84daaf49abed142a373b6e5c59a22dc6560dc refs/heads/master
The bundle requires these 1 ref
9a466c572fe88b195efd356c3f2bbeccdb504102 second commit
../commits.bundle is okay
```

If the bundler had created a bundle of just the last two commits they had done, rather than all three, the original repository would not be able to import it, since it is missing requisite history. The `verify` command would have looked like this instead:

```
$ git bundle verify ../commits-bad.bundle
error: Repository lacks these prerequisite commits:
error: 7011d3d8fc200abe0ad561c011c3852a4b7bbe95 Third commit - second repo
```

However, our first bundle is valid, so we can fetch in commits from it. If you want to see what branches are in the bundle that can be imported, there is also a command to just list the heads:

```
$ git bundle list-heads ../commits.bundle
71b84daaf49abed142a373b6e5c59a22dc6560dc refs/heads/master
```

The `verify` sub-command will tell you the heads as well. The point is to see what can be pulled in, so you can use the `fetch` or `pull` commands to import commits from this bundle. Here we'll fetch the `master` branch of the bundle to a branch named `other-master` in our repository:

```
$ git fetch ../commits.bundle master:other-master
From ../commits.bundle
 * [new branch]      master     -> other-master
```

Now we can see that we have the imported commits on the `other-master` branch as well as any commits we've done in the meantime in our own `master` branch.

```
$ git log --oneline --decorate --graph --all
* 8255d41 (HEAD, master) Third commit - first repo
| * 71b84da (other-master) Last commit - second repo
| * c99cf5b Fourth commit - second repo
| * 7011d3d Third commit - second repo
|/
* 9a466c5 Second commit
* b1ec324 First commit
```

So, `git bundle` can be really useful for sharing or doing network-type operations when you don't have the proper network or shared repository to do so.