tar — Archives & Tarballs

`tar` bundles many files into one archive — a *tarball* — and optionally compresses it. The command itself is simple; what trips people up is the flag soup (`-czf` vs `-xjf`), the fact that compression is a separate layer from the archive, and the classic "tar bomb" that sprays files across your current directory. This sheet covers the combinations you actually use, plus a few lesser-known options that make extracting safer.

On This Page

The mental model

`tar` = *tape archive*. It concatenates files (with their paths and permissions) into a single `.tar` stream. **Compression is a separate step** layered on top — `.tar.gz`, `.tar.xz`, and so on. The letters pick the job:

```
c = create      x = extract      t = list (table of contents)
f = file …      v = verbose      z = gzip   j = bzip2   J = xz
```

Read `-czf` as "**c**reate a g**z**ipped **f**ile" and `-xf` as "e**x**tract this **f**ile." Modern GNU tar auto-detects the compression when extracting, so you rarely need the `z` / `j` / `J` on the way out.

Create an archive

```
tar -cf  archive.tar     files/        # bundle only, no compression
tar -czf archive.tar.gz  files/        # gzip   (.tgz)  — fast, universal
tar -cjf archive.tar.bz2 files/        # bzip2  (.tbz)  — smaller, slower
tar -cJf archive.tar.xz  files/        # xz     (.txz)  — smallest, slowest
tar --zstd -cf archive.tar.zst files/  # zstd — fast with a great ratio (modern)
tar -caf archive.tar.gz  files/        # -a: pick compression from the extension
```

Add `-v` to watch the file names scroll by. Reach for `-a` (auto) when you'd rather name the output by extension than remember which letter is which.

Inspect before extracting

Make this a habit — it's the cheapest way to avoid a mess:

```
tar -tf  archive.tar.gz                # list every member, no extraction
tar -tvf archive.tar.gz                # long listing: perms, size, mtime
tar -tf  archive.tar.gz | head         # just the first entries
```

If the listing shows files at the top level (not tucked under one folder), extracting will scatter them into your current directory — a "tar bomb." List first, or extract with `--one-top-level` (below).

Extract

```
tar -xf archive.tar.gz                      # extract here (compression auto-detected)
tar -xvf archive.tar.gz                     # …and print what's being written
tar -xf archive.tar.gz -C /path/to/dir      # extract INTO an existing directory
tar -xf archive.tar.gz --one-top-level      # into a new folder named after the archive
tar -xf archive.tar.gz --one-top-level=out  # into a new folder you name
tar -xf archive.tar.gz dir/file.txt docs/   # only these members (paths as shown by -t)
tar -xf archive.tar.gz --strip-components=1  # drop the leading path component
```

`--one-top-level` (GNU tar) is the antidote to tar bombs: everything lands inside a single new directory no matter how the archive was built. `--strip-components=N` is handy when a tarball wraps everything in a top folder you don't want (common with GitHub source tarballs).

The flags that matter

Flag	Does
`-c`	create an archive
`-x`	extract
`-t`	list contents (table of contents)
`-f FILE`	operate on `FILE` — almost always required
`-v`	verbose — name each member as it's processed
`-z` / `-j` / `-J`	compress with gzip / bzip2 / xz
`--zstd`	compress with zstd
`-a`	choose compression from the file extension (on create)
`-C DIR`	change to `DIR` first (position-sensitive)
`-p`	preserve permissions (the default when root)
`--exclude=PATTERN`	skip matching paths (repeatable)
`--strip-components=N`	remove `N` leading path parts on extract
`--one-top-level[=DIR]`	extract into a single new directory

Compression formats

Extension	Create flag	Trade-off
`.tar.gz` / `.tgz`	`-z`	fast, everywhere — the safe default
`.tar.bz2` / `.tbz`	`-j`	~10–15% smaller than gzip, noticeably slower
`.tar.xz` / `.txz`	`-J`	smallest, slowest; great for release artifacts
`.tar.zst`	`--zstd`	near-gzip speed, near-xz ratio; needs a recent tar
`.tar`	*(none)*	no compression — required for `-r` / `-u` append

Handy recipes

```
# Back up a project, skipping junk
tar -czf backup.tar.gz --exclude='*.log' --exclude=node_modules project/

# Stream one file out of an archive to stdout (no extraction)
tar -xOf archive.tar.gz path/to/file.txt

# Archive from an explicit list of paths
tar -czf set.tar.gz -T files.txt

# Add a file to an existing UNcompressed archive
tar -rf archive.tar newfile.txt

# Copy a directory tree preserving perms/owners (tar-over-pipe)
tar -cf - -C /src . | tar -xf - -C /dst

# Send a compressed backup over SSH to another host
tar -czf - project/ | ssh user@host 'cat > /backups/project.tar.gz'
```

Gotchas

- **You can't append to a compressed tarball.** `-r` (append) and `-u` (update) need a plain `.tar`. Decompress, append, recompress.
- **`-C` is positional.** It takes effect from where it appears on the line, so `tar -xf a.tgz -C /dst` changes to `/dst` before extracting.
- **`f` is almost always required.** Without `-f FILE`, tar reads/writes the default device (or stdin/stdout) — usually not what you want.
- **GNU vs BSD/macOS tar differ.** macOS ships `bsdtar`: no `--one-top-level`, and some long options vary. `brew install gnu-tar` gives you `gtar` if you need GNU behaviour.
- **Leading slashes are stripped on create.** GNU tar stores absolute paths as relative (a safety feature); `-P` keeps them literal — rarely what you want, and risky on extract.
- **`-z` / `-j` / `-J` on extract are optional but harmless.** Modern tar sniffs the format; you only *need* the letter with very old tars or non-seekable streams.