Bun - Blazlingly Fast JavaScript/TypeScript

R. Mark Volkmann
Object Computing, Inc.
https://objectcomputing.com
mark@objectcomputing.com
@mark_volkmann

OBJECT COMPUTING
YOUR OUTCOMES ENGINEERED

Slides at **https://github.com/mvolkmann/talks/**

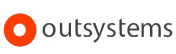
Titanium Sponsors

Platinum Sponsors

Gold Sponsors

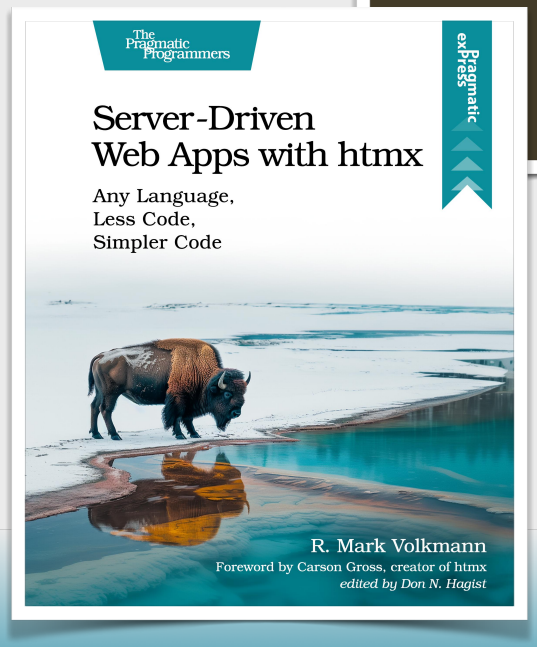
Speaker Dinner

Friends of KCDC



About Me

- Partner and Distinguished Software Engineer at Object Computing, Inc. in St. Louis, Missouri USA
- 43 years of professional software development experience
- Writer and speaker
- Blog at https://mvolkmann.github.io/blog/
- Author of Manning book “Svelte ... in Action”
- Author of Pragmatic Bookshelf book “Server-driven Web Apps with htmx”



Overview

- Bun includes
  - **JavaScript runtime** - replaces Node
  - **package manager** - replaces npm, pnpm, Yarn, ...
  - **bundler** - replaces Parcel, Rollup, Vite, Webpack, ...
  - **test runner** - replaces Jest, Mocha, ...
- Drop-in replacement for Node and npm
  - supports most Node APIs
- Started in 2022 by Jarred Sumner
- Supports Linux, macOS, and Windows

can start by using Bun only for this

Used For

- HTTP servers
- Web framework backends
  - ex. Astro, Elysia, Express, Hono, Next.js, Nuxt, Remix, and SvelteKit
- Alternative to shell scripts
- Anything JavaScript can be used for

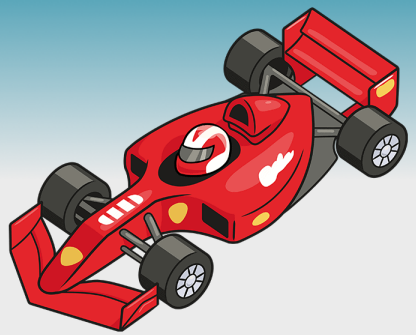
The Name

- Three meanings
  - a friend suggested it because she has a bunny named “Bun”
  - it’s a bundling of the JavaScript ecosystem
  - it’s a bundler

Benefits

- Significantly better **performance**
- Simplifies JS/TS **tooling**
- Native **hot reloading**
  - loads changes without losing state
- Supports both **CommonJS** and **ESM**, even in same file
  - `require` and `import` keywords
- Supports **TypeScript** out of the box
- Built-in support for Jest-compatible **unit tests**
- Better support for standard **Web APIs**
- Supports **JSX/TSX** for HTML generation
- Built-in support for **SQLite**
- **Bun-specific APIs** that are faster than Node alternatives
  - but prevents running in Node

Performance



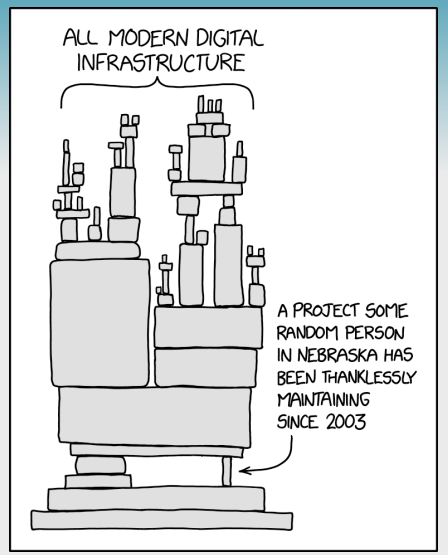
- **Run Hello World**: 4x Node
- **Build and run TypeScript**: 4x esbuild, 15x tsx, 43x tsc + Node
- **Install packages**: 17x pnpm, 29x npm, 33x Yarn
- **Run npm scripts**: `bun run` is 5x `npm run`
- **Run tests**: 5x Vitest, 8x Jest + SWC, 20x Jest + Babel
- **File I/O**: writes 3x Node, reads 10x Node
- **HTTP requests per second**: 4x Node
- **Websocket messages per second**: 5x Node
- **Run SQLite queries**: 2x Deno, 4x Node

"makes other test runners look like test walkers"

SWC is Speedy Web Compiler

Underpinnings

- Built on JavaScriptCore (JSC) used by Safari rather than V8 used by Chrome
  - provides faster startup time and lower memory usage
- Mostly implemented in Zig
  - systems programming language that competes with C++ and Rust
  - Zig 60%, C++ 24%, JavaScript 6%, TypeScript 5%, C 3%, other 2%



Getting Started

- To install on macOS, Linux, and WSL
  - **`curl -fsSL https://bun.sh/install | bash`**
- See docs for other options
- To see a list of bun commands, enter **`bun`** or **`bun --help`**
- For help on a specific command, enter **`bun *command* --help`**
- To upgrade, enter **`bun upgrade`**
- To start a REPL, enter **`bun repl`**

Running npm Packages

- Node **`npx`** command runs a local or remote npm package
  - typically used for remote packages
  - canonical example is **`npx cowsay Hello, KCDC!`**
  - code to run is specified by **`package.json bin`** property
- Bun equivalent is **`bunx`** which is much faster
  - an alias for **`bun x`**
- Try both to see how much faster it is!

Projects ...

- To create a project that uses Bun,
  create a directory, cd to it, and enter **`bun init`**
- Creates the following
  - **`README.md`** - contains basic information about using Bun
  - **`package.json`** - describes dependencies, scripts, and more
  - **`tsconfig.json`** - configures TypeScript
  - **`node_modules`** - directory that holds installed dependencies
  - **`bun.lockb`** - binary equivalent of **`package-lock.json`** for performance
  - **`index.ts`** - entry point source file
- To run, enter **`bun run index.ts`**

initial dependencies are only **`@types/bun`** and **`typescript`**

typically want to create **`src`** directory and move this file there

... Projects

- Can also create a project that use a specific framework
- Enter **`bun create`** ***`template`***
  - where ***`template`*** is **`elysia`**, **`hono`**, **`react-app`**, **`svelte`**, **`vue`**, and more

Package Manager

- Can use Bun for package management
  even if not using it for a JavaScript runtime
  - much faster
  - downloaded packages are stored in **`~/.bun/install/cache`**
  - linked from projects rather than copying

details differ by platform

include **`-d`** for development dependencies

- To add a dependency, enter **`bun add` *`dependency`***
- To remove a dependency, enter **`bun remove` *`dependency`***
- To install all dependencies, enter **`bun install`**
- Tracks dependencies in binary **`bun.lockb`** file

Linting

- Not provided by Bun, but can use ESLint
- Enter **`npm init @eslint/config`**
  to create **`.eslintrc.json`**
  - will ask questions
- Add following script in **`package.json`**
  - **`"lint": "eslint 'src/**/*.{css,html,ts,tsx}'"`**
- To lint project code, enter **`bun lint`**

add **`script`** property
if not already present

Formatting

- Not provided by Bun, but can use Prettier
- Enter **`bun add -d prettier`**
- Create **`.prettierrc`** file
- Add following script in **`package.json`**
  - **`"format": "prettier --write 'src/**/*.{css,html,ts,tsx}'"`**
- To format project code, enter **`bun format`**

```
{
  "arrowParens": "avoid",
  "bracketSpacing": false,
  "singleQuote": true,
  "trailingComma": "none"
}
```

add **`script`** property if not already present

Bundling

- Can bundle all project files and dependencies into a single **`.js`** file
- Add following scripts in **`package.json`**

```
"scripts": {
  "bundle": "bun build src/index.ts --minify --outdir build",
  "clean": "rm -rf build",
  "start": "bun run build/index.js"
}
```

- To create bundle, enter **`bun bundle`**
- To execute bundle, enter **`bun start`**

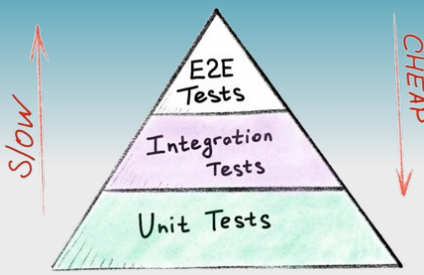
example to try

src/index.ts

```
import { formatDistance } from 'date-fns';

const now = new Date();
const birthday = new Date(now.getFullYear(), 3, 16);
const distance = formatDistance(now, birthday);
console.log(distance, 'until your birthday');
```

Unit Tests ...



- Bun has built-in support for Jest-compatible unit tests
- Basic example

```
export function add(n1: number, n2: number): number {
  return n1 + n2;
}
```
src/math.ts

```
import {expect, test} from 'bun:test';
import {add} from './math';

test('add', () => {
  expect(add(2, 2)).toBe(4);
});
```
src/math.test.ts

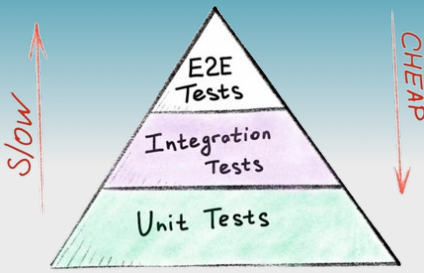
can also import **describe**, **beforeAll**, **beforeEach**, **afterAll**, and **afterEach**

change **test** to **test.skip** to temporarily skip running

- Object returned by **expect** supports 39 matcher methods

summarized at https://bun.sh/docs/test/writing#matchers

... Unit Tests



- To run all unit tests in project, including in subdirectories
  - **`bun test`**
- To only run tests whose name matches a given pattern
  - add **`--test-name-pattern`** ***`pattern`***
- To run only a specific test, change **`test`** to **`test.only`**
  - run with **`bun test --only`**
- To automatically rerun tests when code changes are saved
  - **`bun --watch test`**
- By default tests timeout and are considered failed after five seconds
  - to change, **`add --timeout`** ***`seconds`***

Importing Files

- Bun can import files in several formats including JSON and plain text

haiku.txt

```
Out of memory.
We wish to hold the whole sky,
But we never will.
```

```
import haiku from './haiku.txt';
const lines = haiku.split('\n');
for (const line of lines) {
  const words = line.split(' ');
  console.log(
    `${words.length} words: ${line}`
  );
}
```

dogs.json

```
[
  {
    "breed": "Whippet",
    "name": "Comet"
},
  {
    "breed": "German Shorthaired Pointer",
    "name": "Oscar"
  }
]
```

```
import dogs from './dogs.json';
for (const dog of dogs) {
  console.log(
    `${dog.name} is a ${dog.breed}.`
  );
}
```

Bun.serve

- Can implement basic HTTP servers

index.ts

```
const server = Bun.serve({
  port: 3000,
  fetch(req) {
    const {method} = req;
    const {pathname} = new URL(req.url);
    // Can customize response based on method and pathname.
    return new Response('Hello, World!');
  },
  error(err) {
    console.error(err);
    return new Response(err);
  },
  // optional to support HTTPS
  tls: {
    cert: Bun.file('./cert.pem'),
    key: Bun.file('./key.pem')
  }
});

console.log('Server started on port', server.port);
```

browse localhost:3000

of course these files must exist

package.json

```
"scripts": {
  "dev": "bun --watch index.ts",
  "start": "bun index.ts"
}
```

to start server, enter **bun dev** or **bun start**

--watch restarts server

--hot loads changes without restarting server to avoid losing state

HTTP Libraries

- Elysia and Hono are popular alternatives to Express that simplify implementing HTTP servers
  - see https://elysiajs.com/ and https://hono.dev/
  - both are much faster than Express
  - Elysia is slightly faster than Hono
  - Elysia only runs in Bun; Hono runs in any JS runtime

Hono

- Hono route methods are passed a **`Context`** object
  - used to get request data and create responses
  - can use Zod (https://zod.dev/) to validate request

Action	Code
get value of request header	`c.req.header('Some-Name')`
get value of path parameter	`c.req.param('some-name')`
get value of query parameter	`c.req.query('some-name')`
get value of text body	`const text = await c.req.text();`
get `FormData` from body	`const formData = await c.req.formData();`
get property from `formData`	`const value = (formData.get('property') as string) || '';`
get value of JSON body	`const object = await c.req.json();`

Action	Code
set value of response header	`c.header('Some-Name', 'some value');`
set status code	`c.status(someCode);`
return text response	`return c.text('some text');`
return JSON response	`return c.json(someObject);`
return HTML response	`return c.html(someHTML);`
return "Not Found" error	`return c.notFound();`
redirect to another URL	`return c.redirect('someURL');`

Hono Server ...

- Open dev/hono/hono-demos/dogs-demo in VS Code
- cd to that directory and enter **`bun run dev`**
- Browse localhost:3000
- Use Thunder Client in VS Code to send GET, POST, PUT, and DELETE requests
  - refresh browser after each

index.ts

```
import {type Context, Hono} from 'hono';
import {serveStatic} from 'hono/bun';
import dogRouter from './dog-router';

const app = new Hono();

// Serves static files from public directory.
app.use('/*', serveStatic({root: './public'}));

app.get('/', (c: Context) => c.redirect('/dogs'));

app.route('/dogs', dogRouter);

export default app;
```

Dogs I Know

- Comet is a Whippet.
- Oscar is a German Shorthaired Pointer.

... Hono Server ...

dog-router.tsx

```
import {type Context, Hono} from 'hono';

const router = new Hono();

interface NewDog {
  name: string;
  breed: string;
}

interface Dog extends NewDog {
  id: number;
}

let lastId = 0;

// The dogs are maintained in memory.
const dogMap: {[id: number]: Dog} = {};

function addDog(name: string, breed: string): Dog {
  const id = ++lastId;
  const dog = {id, name, breed};
  dogMap[id] = dog;
  return dog;
}

addDog('Comet', 'Whippet');
addDog('Oscar', 'German Shorthaired Pointer');
```

```
// This gets all the dogs as either JSON or HTML.
router.get('/', (c: Context) => {
  const accept = c.req.header('Accept');
  if (accept && accept.includes('application/json')) {
    return c.json(dogMap);
  }

  const dogs = Object.values(dogMap).sort((a, b) =>
    a.name.localeCompare(b.name)
  );
  const title = 'Dogs I Know';
  return c.html(
    <html>
      <head>
        <link rel="stylesheet" href="/styles.css" />
        <title>{title}</title>
      </head>
      <body>
        <h1>{title}</h1>
        <ul>
          {dogs.map((dog: Dog) => (
            <li>
              {dog.name} is a {dog.breed}.
            </li>
          ))}
        </ul>
      </body>
    </html>
  );
});
```

using JSX

styles.css

```
body {
  font-family: sans-serif;
}
```

tsconfig.json

```
{
  "compilerOptions": {
    ...
    "jsx": "react-jsx",
    "jsxImportSource": "hono/jsx"
  }
}
```

required to use JSX

... Hono Server

```ts
// This gets one dog by its id as JSON.
router.get('/:id', (c: Context) => {
  const id = Number(c.req.param('id'));
  const dog = dogMap[id];
  return dog ? c.json(dog) : c.notFound();
});

// This creates a new dog.
router.post('/', async (c: Context) => {
  const data = (await c.req.json())
    as unknown as NewDog;
  const dog = addDog(data.name, data.breed);
  c.status(201); // Created
  return c.json(dog);
});
```

```ts
// This updates the dog with a given id.
router.put('/:id', async (c: Context) => {
  const id = Number(c.req.param('id'));
  const data = (await c.req.json()) as unknown as NewDog;
  const dog = dogMap[id];
  if (dog) {
    dog.name = data.name;
    dog.breed = data.breed;
  }
  return dog ? c.json(dog) : c.notFound();
});

// This deletes the dog with a given id.
router.delete('/:id', async (c: Context) => {
  const id = Number(c.req.param('id'));
  const dog = dogMap[id];
  if (dog) delete dogMap[id];
  return dog ? c.text('') : c.notFound();
});

export default router;
```

Elysia

- Elysia route methods are passed a **`Context`** object
  - used to get request data and create responses
- Elysia provides its own library for validating request
  - see https://elysiajs.com/essential/schema.html#type

Action	Code
get value of request header	`c.headers['some-name']`
get value of path parameter	`c.params['some-name']`
get value of query parameter	`c.query['some-name']`
get value of text body	`const text = c.body;`
get form data from body	`const formData = c.body;`
get property from `formData`	`const value = formData.some-name;`
get value of JSON body	`const object = c.body;`

Action	Code
set value of a response header	`c.header('Some-Name', 'some value');`
set status code	`c.status(someCode);`
return text response	`return 'some text';`
return JSON response	`return someObject;`
return HTML response	`return someJSX;`
return "Not Found" error	`set.status = 404; return 'Not Found';`
redirect to another URL	`set.redirect = 'someURL'`

An Elysia version of the Hono app we just saw is at https://github.com/mvolkmann/elysia-demos/tree/main/dogs-demo

Environment Variables

- Bun has built-in support for **`.env`** files
- Access environment variables with **`process.env.NAME`** or **`Bun.env.NAME`**

.env

```
NAME=Mark
GREETING="Hello, ${NAME}!"
```

env.test.ts

```
import {expect, test} from 'bun:test';

test('environment variables', () => {
  expect(process.env.NAME).toBe('Mark');
  expect(Bun.env.GREETING).toBe('Hello, Mark!');
});
```

run test with
`bun test`

Executable Files

- Bun can compile to an executable file
- Add script in **`package.json`**

```
"compile": "bun build ./index.ts --outfile birthday --compile",
```

executable

- Enter **`bun compile`**
- This example produces 80.4 MB executable

index.ts

```
#!/usr/bin/env bun
import { formatDistance } from "date-fns";

const now = new Date();
const birthday = new Date(now.getFullYear(), 3, 16);
const distance = formatDistance(now, birthday);
console.log(distance, "until your birthday");
```

library is included in executable

Properties & Methods

- The **`Bun`** global object has many properties and methods
- Particularly useful methods include
  - **`Bun.nanoseconds()`** returns nanoseconds since bun process started
  - **`Bun.sleep(ms)`** returns a **`Promise`** that resolves after **`ms`**
  - **`Bun.sleepSync(ms)`** blocks for **`ms`**
  - **`Bun.deepEquals(obj1, obj2 [, strict])`** recursively compares two objects

see examples on next slide

Bun.deepEquals

index.test.ts

```ts
import { expect, test } from "bun:test";

test("arrays deep equal", () => {
  const a1 = [1, [2, 3]];
  const a2 = [1, [2, 3]];
  expect(Bun.deepEquals(a1, a2)).toBe(true);

  // Trailing undefined elements are ignored
  // unless using strict mode.
  const a3 = [1, [2, 3], undefined];
  expect(Bun.deepEquals(a1, a3)).toBe(true);
  const strict = true;
  expect(Bun.deepEquals(a1, a3, strict)).toBe(false);
});

test("objects deep equal", () => {
  const v1 = { a: 1, b: { c: 3, d: 4 } };
  const v2 = { a: 1, b: { c: 3, d: 4 } };
  expect(Bun.deepEquals(v1, v2)).toBe(true);

  // Extra properties with undefined values are ignored
  // unless using strict mode.
  const v3 = { a: 1, b: { c: 3, d: 4, e: undefined }, f: undefined };
  expect(Bun.deepEquals(v1, v3)).toBe(true);
  const strict = true;
  expect(Bun.deepEquals(v1, v3, strict)).toBe(false);
});
```

Serializing Objects

- Bun can serialize and deserialize objects, even nested

```
import {serialize, deserialize} from 'bun:jsc';
import {expect, test} from 'bun:test';

test('serialize', () => {
  const dogs = [
    {name: 'Comet', breed: 'Whippet'},
    {name: 'Oscar', breed: 'German Shorthaired Pointer'}
  ];
  const buffer = serialize(dogs); // a SharedArrayBuffer
  const newDogs = deserialize(buffer);
  expect(newDogs).toStrictEqual(dogs);
});
```

Writing & Reading Files

- Bun simplifies writing and reading files

```
import {expect, test} from 'bun:test';

test('write/read file', async () => {
  const filePath = './data.txt';
  const content = 'Hello, World!';
  // The second argument to the write function
  // can be a string, Buffer, file, or
  // the awaited result of a fetch call.
  await Bun.write(filePath, content);

  const file = Bun.file(filePath);
  const actual = await file.text();
  // Other methods on File objects include
  // arrayBuffer, blob, and json.
  expect(actual).toBe(content);
});
```

WebSockets

- Bun simplifies using WebSockets

```
const server = Bun.serve({
  port: 1919,
  fetch(req, server) {
    if (server.upgrade(req)) return;
    return new Response('WebSocket upgrade failed', {status: 500});
  },
  websocket: {
    open(ws) {
      console.log('WebSocket opened');
    },
    message(ws, data) {
      console.log('received:', data);
      if (typeof data === 'string') {
        ws.send(data.toUpperCase());
      }
    },
    close(ws, code, reason) {
      console.log(`WebSocket closed with code ${code} and reason "${reason}"`);
    }
  }
});

console.log('WebSocket server is listening on port', server.port);
```

upgrades HTTP connection to WebSocket connection

could send message to client here

SQLite ...

- Bun supports querying and updating SQLite databases
- Must install SQLite

in macOS, **`brew install sqlite`** adds the **`sqlite3`** command

- Example session to create a database

```
sqlite3 todos.db
sqlite> create table todos(id integer primary key, text string, completed numeric);
sqlite> .schema
CREATE TABLE todos(id integer primary key, text string, completed numeric);
sqlite> insert into todos values('t1', 'cut grass', 0);
sqlite> insert into todos values('t2', 'buy milk', 1);
sqlite> select * from todos;
t1|cut grass|0
t2|buy milk|1
sqlite> .exit
```

... SQLite

sqlite.test.ts

```ts
import {Database} from 'bun:sqlite';
import {expect, test} from 'bun:test';

type Todo = {
  id: number;
  text: string;
  completed: number; // 0 or 1 for SQLite compatibility
};

const db = new Database('todos.db');
const deleteAllTodosPS = db.prepare('delete from todos');
const deleteTodoPS =
  db.prepare('delete from todos where id = ?');
const getTodoQuery =
  db.query('select * from todos where id = ?');
const getAllTodosQuery = db.query('select * from todos;');
const insertTodoQuery = db.query(`
  insert into todos (text, completed)
  values (?, 0) returning id
`);
const updateTodoPS =
  db.prepare('update todos set completed=? where id = ?');
```

```ts
test('sqlite', () => {
  deleteAllTodosPS.run();

  const text = 'buy milk';
  const {id} = insertTodoQuery.get(text) as {id: number};
  expect(id).toBeGreaterThan(0);

  let todos = getAllTodosQuery.all() as Todo[];
  expect(todos.length).toBe(1);
  let [todo] = todos;
  expect(todo.text).toBe(text);

  updateTodoPS.run(1, todo.id);

  todo = getTodoQuery.get(todo.id) as Todo;
  expect(todo.completed).toBe(1);

  deleteTodoPS.run(todo.id);

  todos = getAllTodosQuery.all() as Todo[];
  expect(todos.length).toBe(0);
});
```

Bun Shell

- Bun supports executing shell commands and capturing output

File below is named "**`files`**" with no extension, located in **`PATH`**, and made executable with **`chmod a+x`**

Example usage:
`files --limit 5 --verbose ts`
outputs up to 5 file names with extension **`.ts`** and includes the extensions.

files

```
#!/usr/bin/env bun
import {$} from 'bun';
import {parseArgs} from 'util';

const {values, positionals} = parseArgs({
  args: Bun.argv,
  options: {
    help: {
      type: 'boolean'
    },
    limit: {
      type: 'string'
    },
    verbose: {
      type: 'boolean'
    }
  },
  strict: true,
  allowPositionals: true
});
```

```
const {help, verbose} = values;
if (help) {
  console.log(`
Usage: files [options] <extension>
Options:
  --help: Show help
  --limit: Limit number of files to display
  --verbose: Show the full file name
`);
  process.exit(0);
}
```

```
// Will be NaN if missing or invalid.
const limit = Number(values.limit);
const extension = positionals.at(-1);

let count = 0;
const lines = $`ls *.${extension}`.lines();
for await (const line of lines) {
  if (verbose) {
    console.log(line);
  } else {
    const index = line.lastIndexOf('.');
    console.log(line.substring(0, index));
  }
  count++;
  if (count === limit) break;
}
```

Foreign Function Interface (FFI)

- Bun supports calling functions implemented in many languages including C, C+, Kotlin, Rust, and Zig
- For example, build Zig code below as a library with **`zig build-lib average.zig -dynamic -OReleaseFast`**
- Run with **`bun run index.ts`**

average.zig

```zig
pub export fn average(
    numbers_ptr: [*]const f32,
    len: usize
) f32 {
    var sum: f32 = 0.0;
    const numbers = numbers_ptr[0..len];
    for (numbers) |number| {
        sum += number;
    }
    const float_len: f32 = @floatFromInt(len);
    return sum / float_len;
}
```

index.ts

```ts
import {dlopen, FFIType, ptr, suffix} from 'bun:ffi';

// Open a dynamic library.
const path = `libaverage.${suffix}`;
const lib = dlopen(path, {
  average: {
    args: [FFIType.ptr, FFIType.i32],
    returns: FFIType.f32
  }
});

// Get a reference to the average function.
const average = lib.symbols.average;

// Create and pass a typed array.
const numbers = new Float32Array([1, 2, 3, 4]);
const result = average(ptr(numbers), numbers.length);
console.log('average is', result);
```

Resources

- **Bun home page** - https://bun.sh
- **My blog** - https://mvolkmann.github.io/blog/ (select Bun)
- **Bun Discord server** - click in upper-right of Bun home page

Wrap Up

- Bun has many advantages over Node and related tools
  - performance
  - CommonJS and ESM support
  - built-in support for TypeScript, JSX, SQLite, unit tests, bundling, and creating executables
  - calling functions implemented in other languages (FFI)
- Don't need to be all in ...
  start using as a replacement for npm