Scaling well with others

Technical solutions to some of the problems of a moderately-sized team

A true story

1. *CI server #0*: The build is broken!
2. *Dev #0*: Works for me.
3. *Devs #1, 2, and 3*: Works for us.
4. *Devs #4, 5, and 6*: The build is broken!
5. *CI servers #1 and 2*: Works for us.
6. **`<2 hours of head scratching>`**
7. *Dev #4*: **What compiler version is everyone using?**
8. **`<collective facepalm>`**

Word of mouth is no longer good enough

Possible Solutions

- Be extra sure to tell everyone which compiler version to use
- Send a strongly-worded email
- Send ***MULTIPLE*** strongly-worded emails
- Put it in the wiki
- Don’t rely on the Dumb Human™

Check compiler version at build time (project.hxp)

```
if (environment["haxe_ver"] != "3.4.7") {
    Log.error("Incorrect compiler version, expected 3.4.7");
}
```

Check compiler version at build time (project.xml)

```xml
<!-- Enforce haxe compiler version. -->
<set name="req_haxe_ver" value="3.4.7"/>


<!-- perform the comparison that will be checked -->
<set name="wrong_haxe_ver" value="${haxe_ver} != ${req_haxe_ver}"/>


<error value="Wrong compiler version ${haxe_ver}. Expected ${req_haxe_ver}"
    if="$${wrong_haxe_ver}"/>
```

Haxelibs

Don’t do this

project.xml

```
<haxelib name="libname"/>
```

HXML

```
-lib libname
```

Specify your haxelib versions

project.xml

```xml
<haxelib name="libname" version="1.0"/>
```

HXML

```
-lib libname:1.0
```

More haxelib problems

- Every dev has to run `haxelib` every time we upgrade or add a haxelib
- Must update every CI machine
- Can’t easily take fixes without a new release of the library
- Git versions come with their own problems

Our solution

- Commit haxelibs to the project repository
- Everyone gets updates with **`git pull / svn update`** / etc.
- Hotfixes are easy to patch in
- Benefits to versioning dependencies
    - Avoids network-based build breaks
    - Business continuity
    - Troubleshooting

Project-local haxelib repository

- `$> haxelib newrepo`
- Creates **`.haxelib`** directory in current directory
- haxe and haxelib will use the **`.haxelib`** dir as their haxelib repo

Caveats of a local haxelib repo

- Local repo is only used if `.haxelib` dir is in the dir a command is run from
- Not all haxelibs handle a local repo properly
  - `<setenv name="HAXELIB_PATH" value=".haxelib" />`
- Duplicate copies of haxelibs with multiple projects or checkouts
  - `<haxelib repository="../shared/.haxelib" />`
- Libs with binaries can bloat your repository
- Someone still has to manage it all

Haxelibs that use external tools

- Node.js and packages are a good example
- We put all that in source control too
- Has worked very well for us

Downsides to tools in source control

- Binaries in source control can be problematic
- Some things might assume a global tool install

Haxe Completion Server

Haxe Completion Server

- Serves as a compiler cache
- Cuts our build times by 30%
- `haxe -v --wait 6100`
- `haxe --connect 6100 myproject.hxml`
- `openfl build html5 --connect 6100`

Use the completion server by default

project.hxp

```
haxeflags.push("--connect");
haxeflags.push("6100");
```

project.xml

```
<haxeflag name="--connect" value="6100" />
```

HXML

```
--connect 6100
```

Use the completion server by default

```
$> openfl build html5

Fatal error: exception Failure("Couldn't connect on
127.0.0.1:6100")
```

Tell the user about it

project.hxp

```
Log.info("Connecting to haxe completion server on port 6100\n" + "If you haven't already, open a new terminal and run 'haxe --wait 6100' and re-run your build");
```

project.xml

```
<echo value="Connecting to haxe completion server on port 6100"/>
```

Completion server pain points

- Annoying
- Port conflicts

  -
    ```
    var port = !defines.exists("hxport") ? "6000" :
        defines.get("hxport");
    haxeflags.push(port);
    ```
- CI environment

  -
    ```
    if (!environment.exists("HX_NO_CONNECT")) {
        haxeflags.push("--connect");
        haxeflags.push("6100");
    }
    ```

Questions?