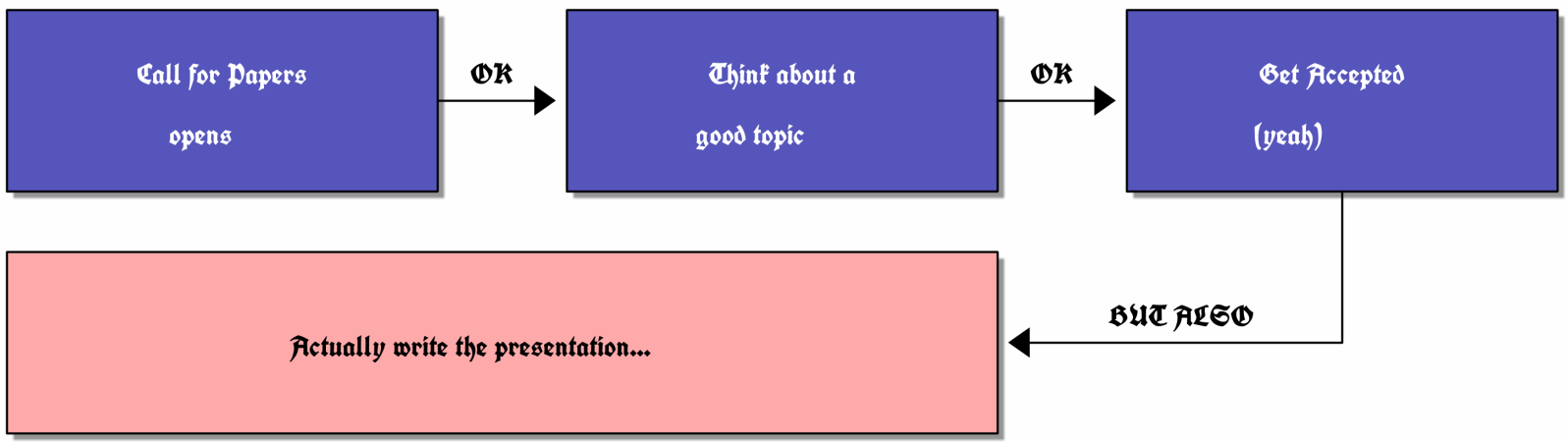
How This Presentation Was Made

Markus Hauck @markus1189

Presentations



Some Problems

- powerpoint/keynote/google slides/...
- but you can't use `git`
- pandoc / LaTeX / ...
- how to include code and pictures?

How It All Started

- Me writing presentation be like:
- fighting graphical editor more than focused on content
- logical step: switch to something that is text based
- how to handle generated pictures
- how to handle code

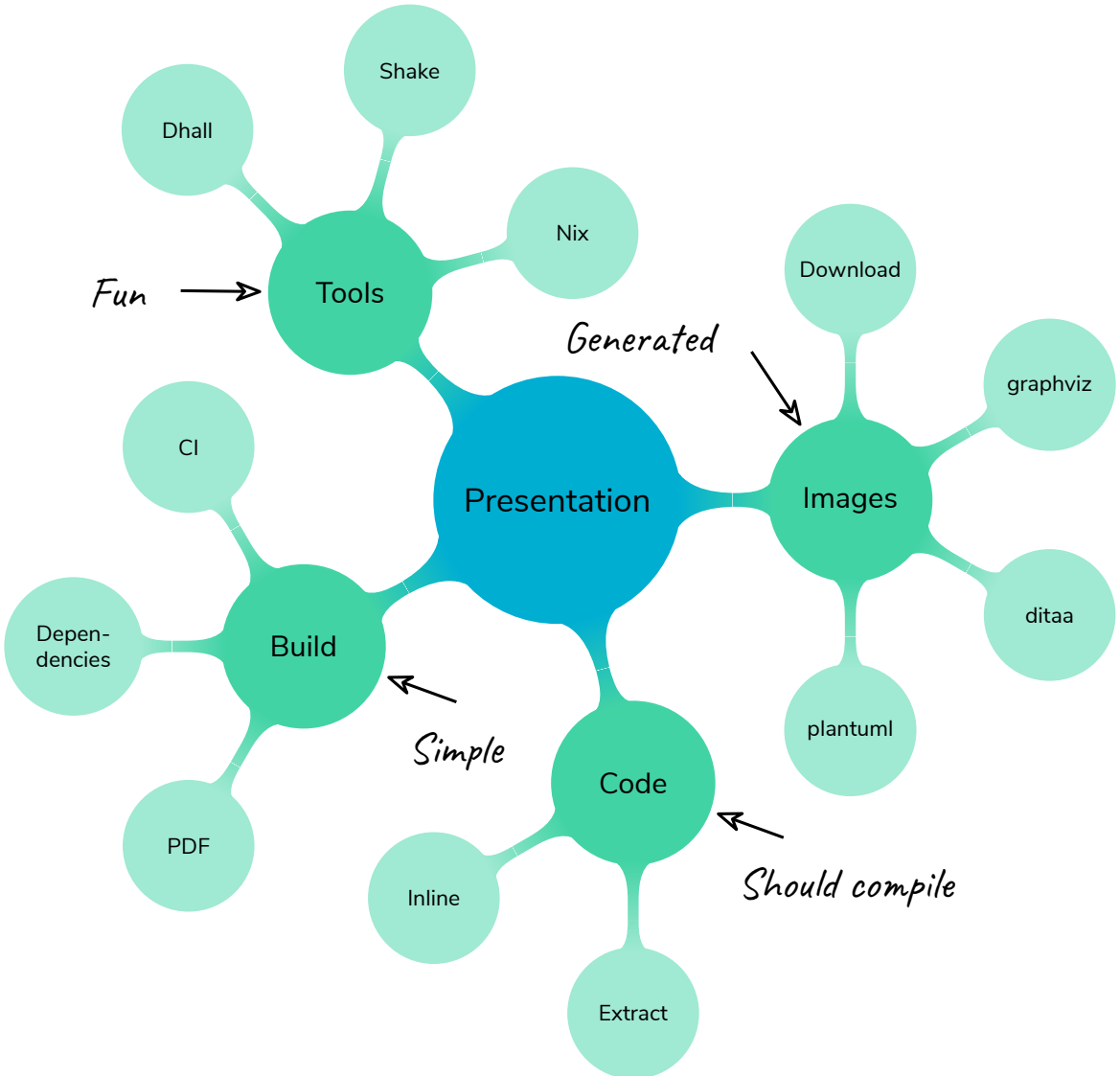
Used Tools Overview

- Nix for system dependencies + build env
- Shake to write a custom build system
- Dhall for “configuration”
- LaTeX for slides
- ditaa, graphviz

Wish List

- version control: use git to track changes
- reproducible: same description for CI and local machine
- single step: **one** command to build presentation
- declarative: generate from description
- checked: source code compiles
- minimal: only re-build what changed

Overview


Shake
Dhall
Nix
Fun
Tools
Generated
Download
graphviz
CI
Presentation
Images
ditaa
Depen-
dencies
Build
plantuml
Simple
Code
PDF
Inline
Should compile
Extract

Tool: Shake

- `shakebuild.com/manual`
- Shake is a Haskell **library** for writing build systems
- “just” a library, rest is up to you
- Shake vs `make` is like `Monad` vs `Applicative`
- integrates well with other libraries and system tools

Shake — Usage

- specify rules to create output from some input
- avoid rebuilds of unchanged things
- call the shake build from your `main`

```haskell
theMain :: IO ()
theMain =
  shakeArgs
    shakeOptions
    (want [buildDir </> "slides.pdf"])
```

Shake Rules

```
1  --  +----------------- file pattern to match
2  --  |
3  --  |       +-------- target path to create
4  --  |       |
5  --  v       v
6  pattern %> \out -> do
7    action1           -- <--\
8    action2           -- <---+- Actions to build 'out'
9    action3           -- <--/
```

Shake Rules

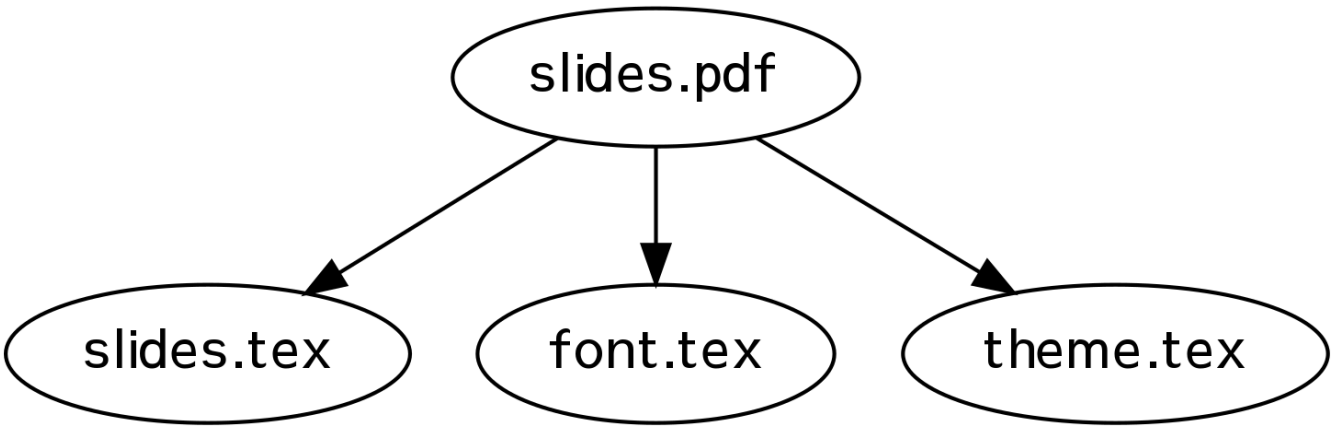
```
1  "*.txt" %> \out -> do
2    putNormal "Debug"
3    cmd "touch" [out]
```

Shake Rules

```
buildDir </> "slides.pdf" %> \out -> do
  let inp = out -<.> "tex"
  need (inp : includedFont : beamerThemes)
  latexmk inp
```

```
latexmk :: FilePath -> Action ()
latexmk inp =
  cmd
    (Cwd (takeDirectory inp) : cmdOpts)
    "latexmk"
    [ "-g"
    , "-shell-escape"
    , "-pdfxe"
    , dropDirectory1 inp
    ]
```

Shake Rules



Shake

- general idea: express any dependencies via Shake rules
- let shake figure out what needs rebuilding
- ensures minimal rebuilds
- you don’t have to worry

Shake

- quick status: use haskell library shake
- get a “build system” for your presentation
- next: source code

Editing Code

- Step 1: Implement your code in a normal project
- Step 2: Wild Copy And Paste Into Presentation
- Step 3: Reformat To Fit Slide
- Step 4: Change Original Source Code
- Step 5: Wild Editing Of Code on Slides
- Step 6: Notice something doesn't make sense

Extract Code

- totally broken: copy & paste
- little better: extract based on lines, still bad
- after edit / formatting / ...they change
- not what we want

Editing Code

- idea: extract source code directly from actual project
- use comments to delimit “snippets”
- write code to extract everything in between

Editing Code

- add comments in the code
- write a small “snippet” file
- let shake automatically extract snippets
- include code snippets in presentation

Annotating Code for Snippets (META)

```
1  --snippet:pdf rule
2  buildDir </> "slides.pdf" %> \out -> do
3    let inp = out -<.> "tex"
4    need (inp : includedFont : beamerThemes)
5    latexmk inp
6  --end:pdf rule
```

Intermezzo: Dhall

A configuration language guaranteed to terminate

- think: lambda calculus for config
- not turing-complete on purpose
- subset can be converted to JSON and YAML
- has **types**!!!11
- can be mapped directly into Haskell types

Dhall Example

```
{ a =
    42
, b =
    [] : Optional Natural
, c = -- comments are possible
    { c1 = "foo" : Text, c2 = "bar", c3 = [ 1, 3, 3, 7 ] }
}
```

Dhall Features

- booleans/integer/naturals
- optional values
- lists
- records
- functions
- strings + interpolation
- unions
- imports
- ...

Dhall To JSON

```
{
  "a": 42,
  "b": null,
  "c": {
    "c1": "foo",
    "c3": [
      1,
      3,
      3,
      7
    ],
    "c2": "bar"
  }
}
```

Dhall To YAML

```
a: 42
b: null
c:
  c1: foo
  c3:
  - 1
  - 3
  - 3
  - 7
  c2: bar
```

Snippet Files — Type

```
{ snippetFile :
    Text
, snippetStart :
    < Search : { term : Text } | End : {} | Start : {} >
, snippetEnd :
    < Search : { term : Text } | End : {} | Start : {} >
}
```

Snippet Files — Haskell

```haskell
data SnippetSrc = SnippetSrc
  { snippetFile :: Text
  , snippetStart :: Addr
  , snippetEnd :: Addr
  } deriving (Show, D.Generic)

instance D.Interpret SnippetSrc
```

Snippet Files — Example

```
1     { snippetFile =
2         "snippets/Snippet"
3     , snippetStart =
4         ./Addr .Start {=}
5     , snippetEnd =
6         ./Addr .End {=}
7     }
8   : ./Snippet
```

Snippet Files — Example

```
1     { snippetFile =
2         "Build.hs"
3     , snippetStart =
4         ./Addr .Search { term = "--snippet:pdf rule" }
5     , snippetEnd =
6         ./Addr .Search { term = "--end:pdf rule" }
7     }
8   : ./Snippet
```

Snippet Files — Extraction

```
1  extractSnippet :: FilePath -> Action String
2  extractSnippet file = do
3    putQuiet ("Extracting from " <> file)
4    need [file]
5    SnippetSrc (T.unpack -> sourceFile) startSearch endSearch <-
6      readDhall file
7    lns <- readFileLines sourceFile
8    let result = findSnippet startSearch endSearch lns
9    if null result
10     then error
11            ("Empty snippet for:\n" <> file <> ":0:")
12     else return (unlines result)
```

Extracting Code

- will always be up to date with the compiling source (yay)
- but we also have to format and maybe check again

Checking Code

- let's tackle checking first
- lots of times: broken code snippets that don't compile
- style errors you would notice in your actual setup
- after extracting a snippet into an includable file
- run linter/compiler/...
- fail building presentation if the command fails

Checking Code

- haskell with hindent + hlint
- scala with sbt and scalafmt
- actually any programming language and linter

Formatting Code

- just another step like linting
- run formatter of choice on the source file
- e.g. format to a width of 55 chars

Snippet Rule — Broken Formatting

```
1  --snippet:hs snippet rule
2  buildDir </> "snippets" </> "*.hs" %> \out -> do
3    snip <- extractSnippet (dropDirectory1 $ out -<.> "snippet")
4    withTempFile $ \temp -> do
5      liftIO (writeFile temp snip)
6      hlint temp
7      hindent temp
8      content <- liftIO (readFile temp)
9      writeFileChanged out content
10 --end
```

Snippet Rule — After Auto-Formatting

```haskell
buildDir </> "snippets" </> "*.hs" %> \out -> do
  snip <-
    extractSnippet
      (dropDirectory1 $ out -<.> "snippet")
  withTempFile $ \temp -> do
    liftIO (writeFile temp snip)
    hlint temp
    hindent temp
    content <- liftIO (readFile temp)
    writeFileChanged out content
```

Snippets — Summary

- automatic snippet extraction, robust
- snippets compile
- snippets are formatted
- no worries over broken code
- confidence in examples

Pictures

- scenario 1: search on the web and download
  - but you will forget from where
  - resize and rotate are manual steps
  - you have to store them in git
- scenario 2: generated from description
  - graphviz graphs
  - ditaa diagrams
  - plantuml diagrams
  - and more...

Downloading Pictures

Picture: me having to do too many manual steps

Downloading Pictures

- use Haskell and Shake to download on demand
- download of the file from the internet
- file that describes from where plus transformations
- transformations performed by imagemagick

Downloading Pictures

```
{
  url = "http://bit.ly/maintain-make-jpg",
  transformations = ["-resize 1101"]
}
```

Downloading Pictures

```
1    [ buildDir </> "images/*" <.> ext
2    | ext <- ["jpg", "png", "gif"]
3    ] |%> \out -> do
4      let inp = dropDirectory1 $ out -<.> "src"
5      need [inp]
6      ImageSrc uri ts <- traced "image-src" (readDhall inp)
7      download downloadResource (TS.unpack uri) out
8      for_ ts $ unit . applyTransformation out
```

Generating Pictures

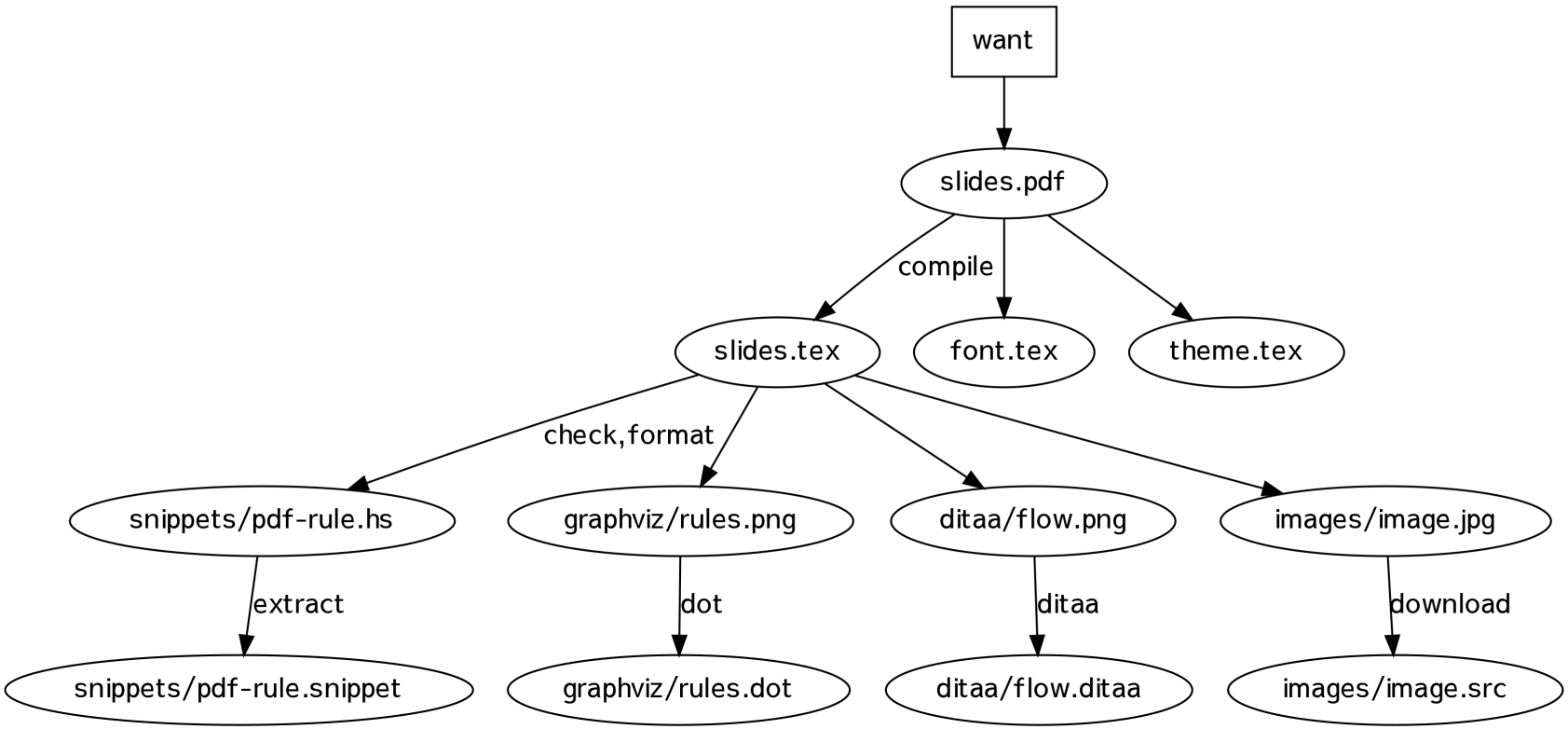
- second scenario: picture is generated
- there is a file that describes it + tool to render
- Steps:
  - write the description file
  - generate graphic
  - include in presentation
  - change description
  - generate graphic
  - include in presentation
  - change description again...

Shake It

- express the dependency as a shake rule

```
1  buildDir </> "graphviz/*.png" %> \out -> do
2    let inp = dropDirectory1 out -<.> "dot"
3    need [inp]
4    graphviz inp out
```

Everything As A Rule



Getting Dependencies

- the missing piece: how to “discover” dependencies?
- all of hackage is available
- parse LaTeX via HaTeX (this time)
- use the pandoc library
- …whatever you need

Extraction from LaTeX

```
commandDeps ::
      [String] -> FilePath -> Action [[FilePath]]
commandDeps cmds file = do
  etex <- liftIO (parseLaTeXFile file)
  case etex of
    Left err ->
      error
        ("Parsing of file " <> file <> " failed: " <>
         show err)
    Right t -> do
      let result =
            map (map T.unpack . mapMaybe cmdArgs . snd) .
            matchCommand (`elem` cmds) $
            t
      return result
```

Develop Environment

- we freely mixed stuff and used lots of tools
  - haskell + libraries
  - imagemagick
  - graphviz
  - ditaa
  - LaTeX plus packages and special font
  - scala, sbt, scalafmt

Continuous Integration via Travis

```
sudo: required

language: haskell

cache:
  directories:
  - ~/.stack

branches:
  except:
    - pdf

before_install:
  - wget -q -O- http://download.fpcomplete.com/ubuntu/fpco.key | sudo apt-key add -
  - echo 'deb http://download.fpcomplete.com/ubuntu/precise stable main' | sudo tee /etc/apt/sources.list.d/fpco
  - sudo apt-get update

  - sudo apt-get install -qq -y stack
  - stack --version

  - sudo apt-get install -qq -y texlive-base texlive-fonts-recommended texlive-latex-extra texlive-latex-recomme
  - latex --version

  - sudo apt-get install -qq -y python-pip
  - pip --version

  - wget 'http://downloads.typesafe.com/scala/2.11.7/scala-2.11.7.tgz' && tar xzf scala-2.11.7.tgz && export PAT
  - scala -version

  - sudo apt-get install -qq -y fort77 libblas3gf libblas-doc libblas-dev liblapack3gf liblapack-doc liblapack-d
  - sudo apt-get build-dep -qq -y r-base r-base-dev
```

Continuous Integration via Travis

```
    - sudo apt-get build-dep -qq -y r-base r-base-dev

    # May result in error during R ./configure
    # checking whether mixed C/Fortran code can be run... configure: WARNING: cannot run mixed C/Fortran code
    # configure: error: Maybe check LDFLAGS for paths to Fortran libraries?
    - sudo mv /usr/lib/libf2c.so /usr/lib/libf2c.so_backup
    - sudo ln -s /usr/lib/libf2c.a /usr/lib/libf2c.so

    - wget 'https://cran.r-project.org/src/base/R-3/R-3.1.3.tar.gz' && tar xzf R-3.1.3.tar.gz
    - (cd R-3.1.3 && ./configure --with-blas --with-lapack --prefix=/usr/local && make)
    - export PATH="$(pwd)/R-3.1.3/bin:$PATH"
    - R --version
    - R --slave --no-save -f R/install-packages.R

    - sudo pip install --upgrade pygments
    - pygmentize -V

  install:
    - mkdir -p "$HOME/texmf/tex/latex"
    - (cd "$HOME/texmf/tex/latex" && wget -O lineno.zip "http://mirrors.ctan.org/macros/latex/contrib/lineno.zip"
    - texhash "$HOME/texmf"
    - (cd "$HOME/texmf/tex/latex" && wget -O minted.zip "http://mirrors.ctan.org/macros/latex/contrib/minted.zip"

    # setup $HOME/bin
    - mkdir -p "$HOME/bin"
    - export PATH="$HOME/bin:$PATH"

    # install latexmk
    - (mkdir -p "install-latexmk" &&
       cd "install-latexmk" &&
```

Continuous Integration via Travis

```
    wget -O latexmk.zip "http://mirrors.ctan.org/support/latexmk.zip" &&
    unzip latexmk.zip &&
    cp latexmk/latexmk.pl "$HOME/bin/latexmk" &&
    chmod +x "$HOME/bin/latexmk")
-   latexmk -version

# install custom design and fonts
- wget -O some-design.zip 'http://some-custom-design_1.0.20140928.zip'
- unzip some-design -d "$HOME/"
- wget -O some-other-design.zip 'http://some-other-custom-design_0.0.20140703.zip'
- unzip some-other-design.zip -d "$HOME/"
- unzip some-fonts.zip -d "$HOME/"

- (wget -O excludeonly.zip "http://mirrors.ctan.org/macros/latex/contrib/excludeonly.zip" && unzip excludeonly
- (wget -O cleveref.zip "http://mirrors.ctan.org/macros/latex/contrib/cleveref.zip" && unzip cleveref.zip -d "
- (wget -O microtype.zip "http://mirrors.ctan.org/macros/latex/contrib/microtype.zip" && unzip microtype.zip -

- mkdir -p "$HOME/.texmf-var"
- mkdir -p "$HOME/.texmf-config/updmap.d"

- mv -v "$HOME/texmf/updmap.d" "$HOME/.texmf-config/"
- cat "$HOME/.texmf-config/updmap.d/20tex-fonts.cfg"
- sed -i -e '/DebPkgProvided/d' "$HOME/.texmf-config/updmap.d/20tex-fonts.cfg"
- texhash "$HOME/texmf"

- update-updmap || echo update-updmap failed
- updmap || echo updmap failed

# Diagnosis
- updmap --listmaps
```

Continuous Integration via Travis

```
    - updmap --listmaps | egrep "^Map[[:blank:]]*5" || echo nop
    - kpsewhich --all updmap.cfg

    - stack setup
    - stack build hlint

  script:
    - stack build
    - stack exec thesis

  after_failure:
    - ls -1lhA
    - cat thesis.log
    - cat thesis.blg
    - cat presentation/final/final.log

  after_success:
    - git config --global user.email "travis-ci@travis.org"
    - git config --global user.name "Travis CI"
    - git checkout -b pdf
    - git add -f thesis.pdf
    - git add -f presentation/final/final.pdf
    - git commit -m "$(date --iso-8601) @ $(git rev-list --max-count=1 --abbrev-commit $TRAVIS_BRANCH)" -m "[skip
    - git tag -f -a -m "Compilation on $(date --iso-8601) of commit $(git rev-list --max-count=1 --abbrev-commit $
    - git push --tags -f origin pdf
```

Continuous Integration Madness

- it’s huge and a mess, good luck maintaining this
- OS specific, your own setup vs travis
- not reproducible at all
- very brittle

The One Command Lie

- you just have to run this **one** command
- it's mostly a lie
- with nix, you can *actually* achieve that!
- perfect: use it in ".travis.yml" as well as every pc

Nix

- `https://nixos.org/nix/`

  Nix is a powerful package manager for Linux and other Unix systems that makes package management reliable and reproducible.

Continuous Integration Made Easy

```
language:
  - nix

script:
  - "./Build.hs"
```

Executing Our Shake Build

```
1 #! /usr/bin/env nix-shell
2 #! nix-shell shell.nix -i "runhaskell
  ↪  --ghc-arg=-threaded --ghc-arg=-Wall"
3 #! nix-shell --pure
```

The Nix File (Simplified)

```
1  { pkgs ? import <nixpkgs> {}}: with pkgs;
2
3  mkShell {
4    buildInputs = [
5      (texlive.combine {
6        inherit (texlive)
7        beamer;
8      })
9      pythonPackages.pygments
10     graphviz
11     imagemagick
12     (ghc.withPackages (p: with p; [shake dhall]))
13   ];
14 }
```

Only LaTeX

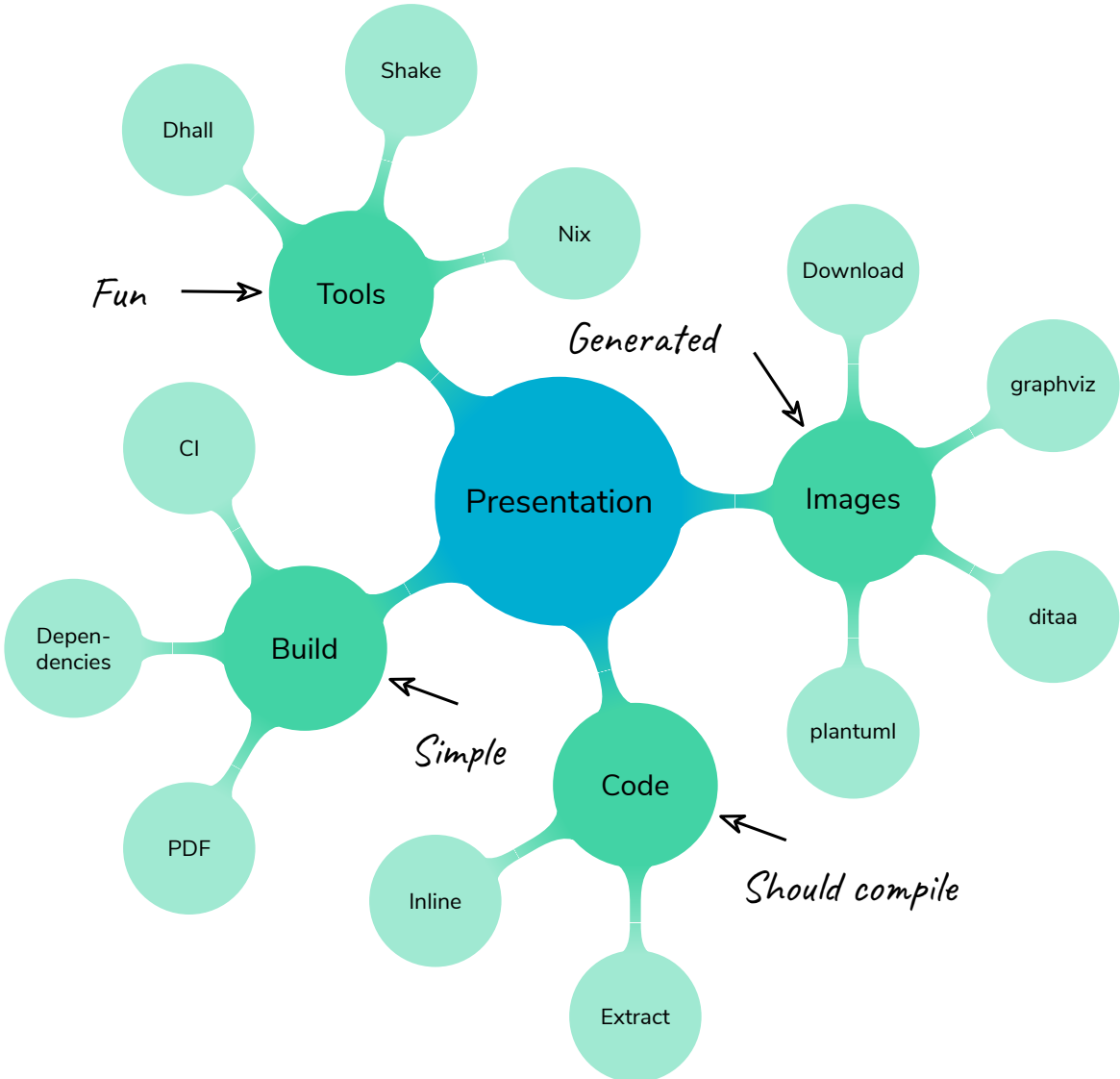
- all of this is not specific to LaTeX
- other: pandoc, reveal.js, ...
- e.g. download reveal.js automatically
- use pandoc to analyze the markdown

I Want To Use This

- github.com/markus1189/how-this-presentation
- you need `Nix`, but that's it

Thanks for your attention

Markus Hauck
(@markus1189)


Shake
Dhall
Nix
Fun
Tools
Generated
Download
graphviz
CI
Presentation
Images
Depen-
dencies
Build
ditaa
plantuml
Simple
Code
PDF
Inline
Should compile
Extract