# My Organization's First R Package

## 3 — Package Management

# Finding out what's needed to make the package a success

So you’re making a work package… What’s going to make it work out?

First, it helps if people care about it.

You might have to sell the idea to management and coworkers.

Getting people onboard with the idea can open doors.

## EXAMPLES OF PEOPLE NOT CARING ABOUT IT

Management doesn’t care

Pitch 📦 development in terms that management cares about.

Your coworkers don’t care

Talk about the all the good things.

You don’t care? 😒

I don’t believe it… you’re here.
This is impossible.

# Getting people involved and giving their roles

If you manage to get people excited enough, give them something to do!

*Reporting Functions*

*API Functions*

*Finance Functions*

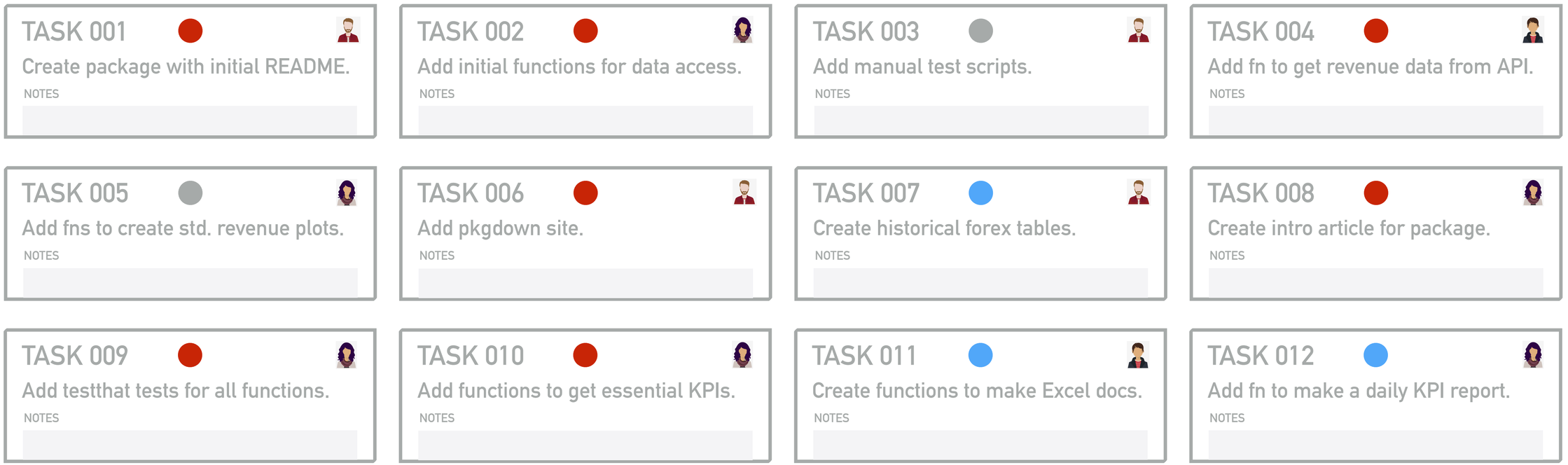
Having these sorts of defined roles can allow for separate focus areas in the package.

# Initial planning and prioritizing

Create an initial set of items to do. Have that information annotated & accessible.

If you have **GitHub** or similar, use Issues, tags, and milestones

Just using **JIRA**? Or **Trello**? That's okay, use that.

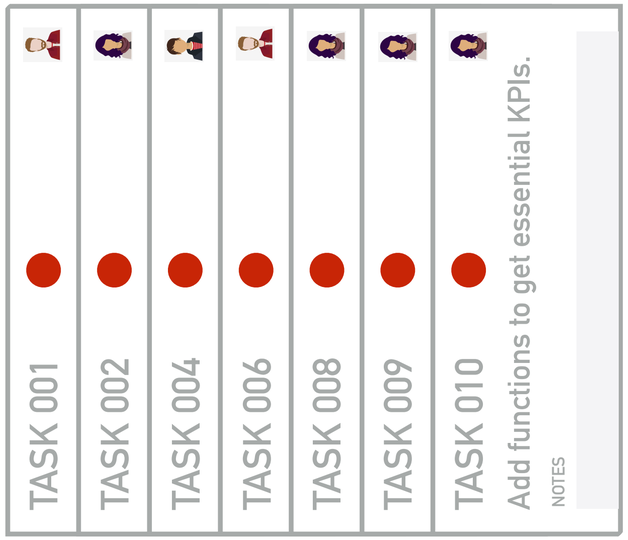


# Initial planning and prioritizing

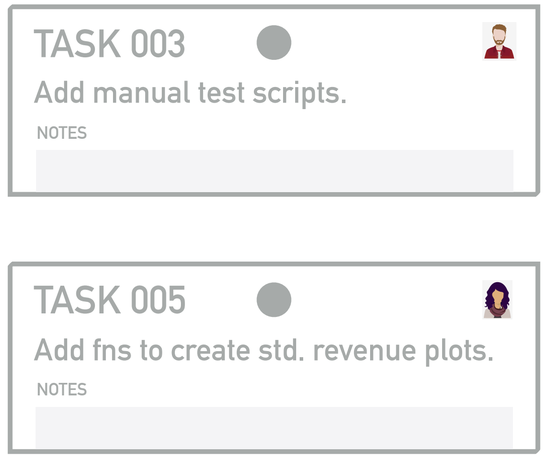
Create an initial set of items to do. Have that information annotated & accessible.

If you have **GitHub** or similar, use Issues, tags, and milestones

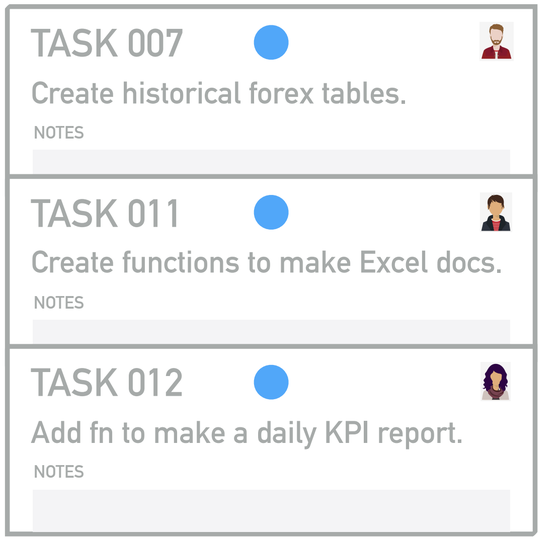
Just using **JIRA**? Or **Trello**? That's okay, use that.




*initial release*




*initial release*
*(stretch goal)*




*second release*

# Hosting the package in the organization

Where do you host the package? This is less obvious in an organization.

This can be in Enterprise versions of **GitHub**, **GitLab**, or **BitBucket**.

This can be in self-hosted versions of **GitHub**, **GitLab**, or **BitBucket**.

Package source could be in an enterprise network location.

```
devtools::build(path = "/path/to/dir")
devtools::install_local("/path/to/dir/
pkgname_pkgversion.tar.gz")
```

The package could be downloadable from an internal website.

The package can be placed in the *System Library* (sometimes by IT).

The package can be hosted in **RStudio Package Manager**.

# The README as the selling point of your package

Having a compelling `README` can get people to use and contribute to the pkg.

Try to keep it pretty brief, saving the details for the `pkgdown` site. → We'll talk about the **pkgdown** site in the next slide!

Some ideas for sections in the `README`:

**Introduction**

**Examples**

**Installation** → Make sure to test this.

**Code of Conduct** → `usethis::use_code_of_conduct()`

**License** → Essential to have in the pkg.
More details coming up.

# A pkgdown site is still insanely useful in an organization

It's a one-stop shop for instructional articles and function documentation.

Use `usethis::use_pkgdown()` to set things up.

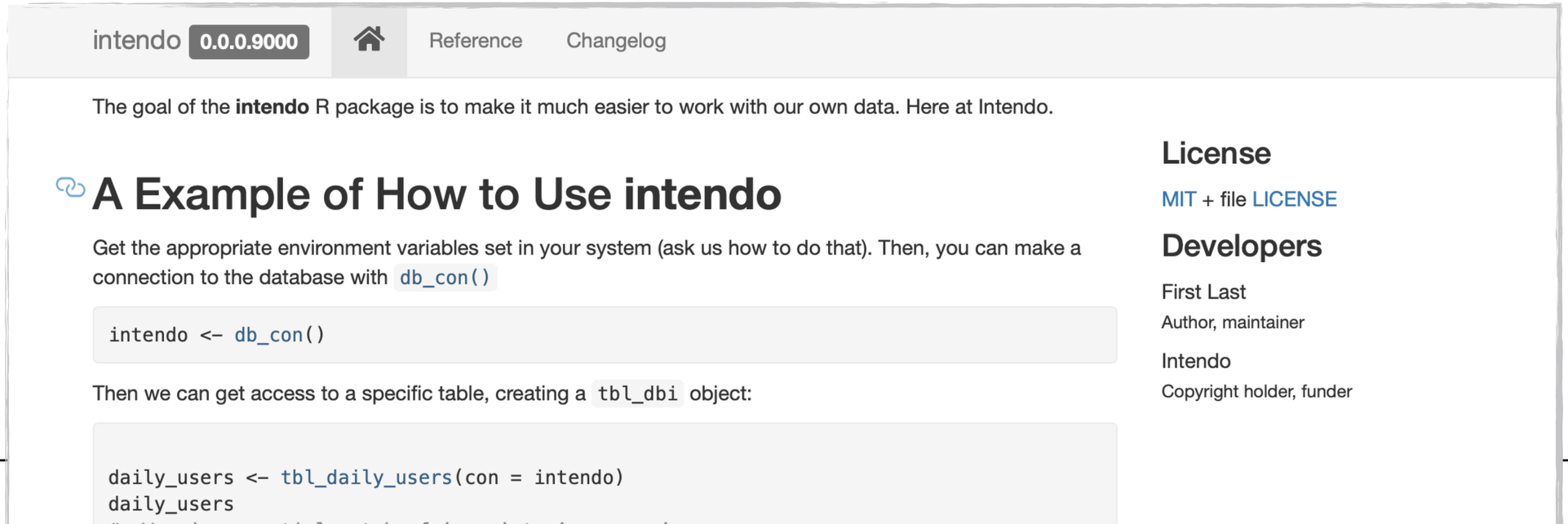
Use `pkgdown::build_site()` for near-instantaneous site generation.

# A pkgdown site is still insanely useful in an organization

It's a one-stop shop for instructional articles and function documentation.

Use `usethis::use_pkgdown()` to set things up.

Use `pkgdown::build_site()` for near-instantaneous site generation.



# Internal documentation should be really useful

Make the help… helpful.

The *Title* is important. It should quickly tell us the use of the fcn.

Describe the `@param` values (the arguments, really) in detail.

**Controversial:** Include docs for unexported functions. And `@noRd`.

Fill in the *Description* and the *Details* sections.

Do use the *Examples* section. Add some reproducible example code.

`@family` and `@seealso` are pretty cool **roxygen** tags.

## Super simple examples as documentation: `tldr`

This is a great idea I haven't seen implemented in **R** packages yet.

`tldr.sh`

(This is a website.)

The proposal is to make a function like `tldr_{pkg_name}()`.

The function will simply print minimal examples of important functions.

## The data dictionary: so useful, yet, we don't see it often enough

Anyone who helps to write a data dictionary will be elevated to hero status.

We do we do it? Document our DB tables like we would document datasets!

In fact, get a `slice()` from every database table and *make a dataset from it*.

Then, follow the usual pattern for documenting datasets.

Then, you'll have a data dictionary!

**USEFUL usethis FUNCTIONS**

`use_data_raw()` `edit_file()` `use_data()`

# Training & education requirements for the package

If the package has a lot to it, there's probably much to teach about.

Teaching ideas:

– lunch & learns

– training workshops

– small group tutorials

– 1-to-1 training

Training material (if available) should be posted somewhere accessible.

# Quality measures and quality control

How do we and others know the package can be trusted? QA/QC helps.

Lots of **testthat** tests; publication of code coverage results.

Lots of manual tests; having an article that discusses these tests.

`R CMD check`. Still a good idea even if **CRAN**'ll never be involved.

Have a QA person test releases and do regular testing on real data.

Have a policy to prioritize and fix bugs that hamper quality of results.

Code reviews and code pairing sessions. Helps to find problems.

# Tracking Issues

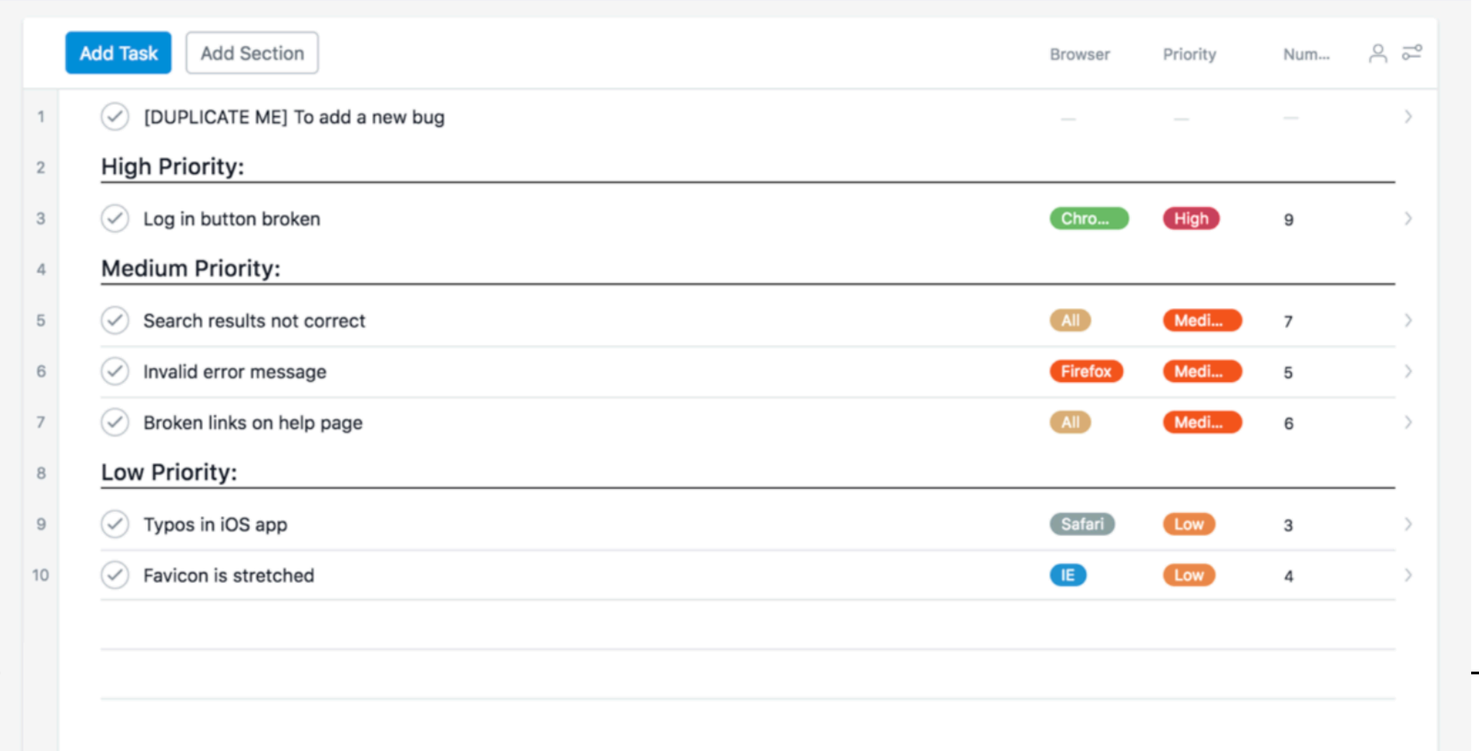
We need one good place for people to create package issues.

Issues are most easily managed in **GitHub**, **GitLab**, or **BitBucket**.

If not using those platforms for development, maybe use **JIRA**?

There are other options too. Like:

**Trello**
**Asana**
**Zoho BugTracker**



# The package should have a license... which one?

You may not think it's important but, one day, it might be. So nip this in the bud.

The `LICENSE` is important to have on Day 1 of the package. (There are legal reasons.)

There might come a day when the package is open-sourced so it's important for that reason.

Even if it'll be closed-source in perpetuity, a license is important. Here are some ideas:

**THE MIT LICENSE**

```
(C) <ORG NAME>. All rights reserved.
```

Use: `www.binpress.com/license-generator/` to generate a license

# Communicating the value of the package

The package will inevitably create a ton of value. Don't keep that a secret.

Use the name in internal data products, during presentations, during discussions. → The name might be part of a larger analytics system, which is also good.

Create an annual report that outlines the state of the the package.

This doesn't have to be big. Just a document that demonstrates the value of the work and where development is heading.

# DISCUSSION

Let’s talk about all of this.

### SOME DISCUSSION POINTS

Did you do any of these things IRL?

1. Discuss the idea of creating an **R** package for an organization.
2. Ask people if they wanted to help with making an **R** package.
3. Created an org package in a solo capacity, introducing it later.

What went well and what didn't?

Do you have any advice from personal experience?

Want to share some tips/tricks w.r.t. package management?

# YOUR TURN (WORKING ON THE intendo PACKAGE)

## 45 minutes

**Open up the package-in-progress at:**

`p_03_intendo/03_intendo.Rproj`

There are four things we should work on:

1. Improve the package `README.Rmd` (add an *Intro* section).
2. Add an example to each function (to the **roxygen** docs).
3. Create a nice **pkgdown** site (edit the `_pkgdown.yml` file to describe the fcns).
4. Add a **testthat** test for each exported function.

**USEFUL FCNS** `use_pkgdown()` `use_testthat()` `use_test()`

# Let's discuss what we worked on

How long did each task take? Which one took the longest?

Were the tests difficult to write? Do you foresee any problems with writing tests that use DB data?

Have you ever created a **pkgdown** site before? Was there anything surprising about the result?