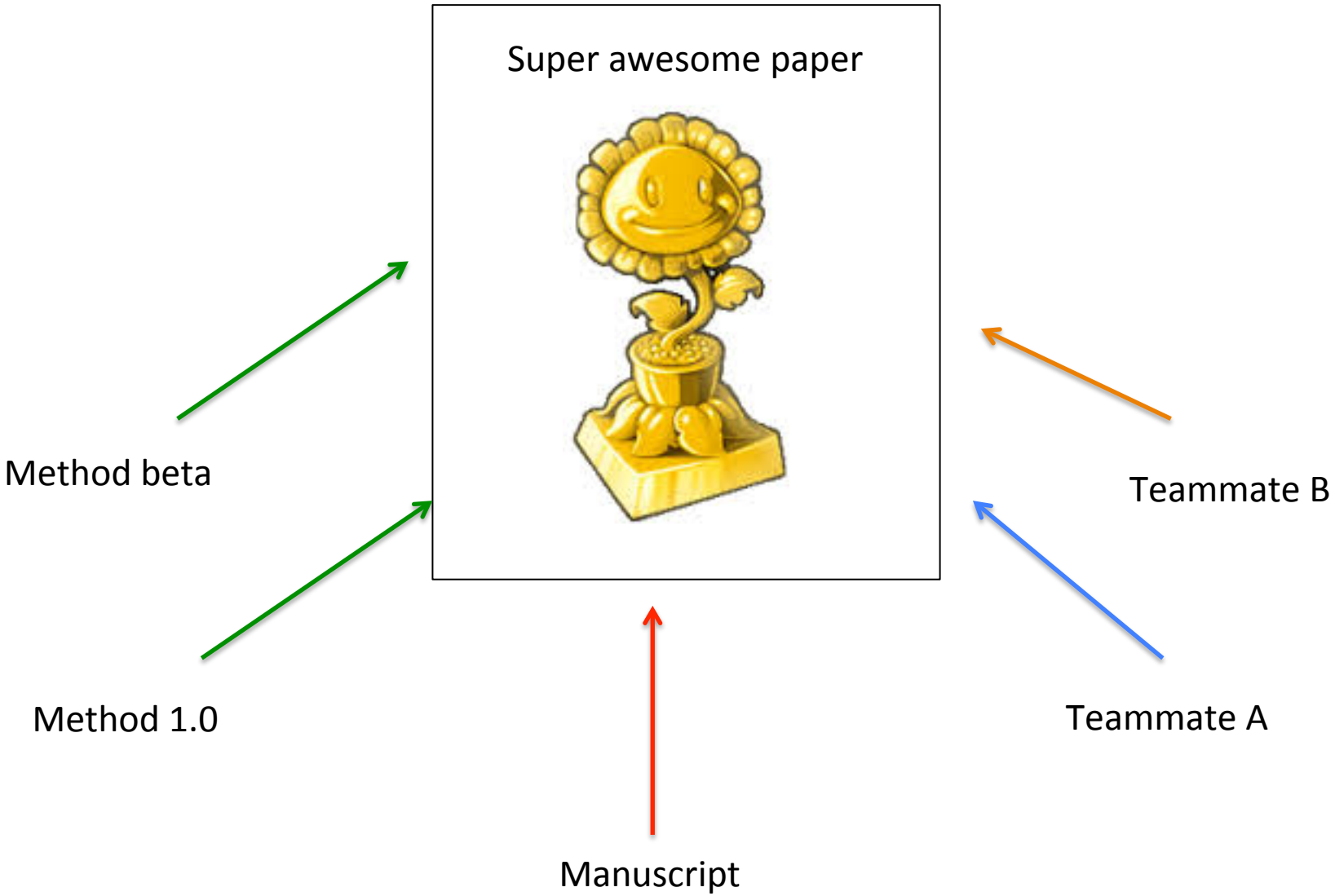
Collaborating on GitHub

Stephens lab meeting

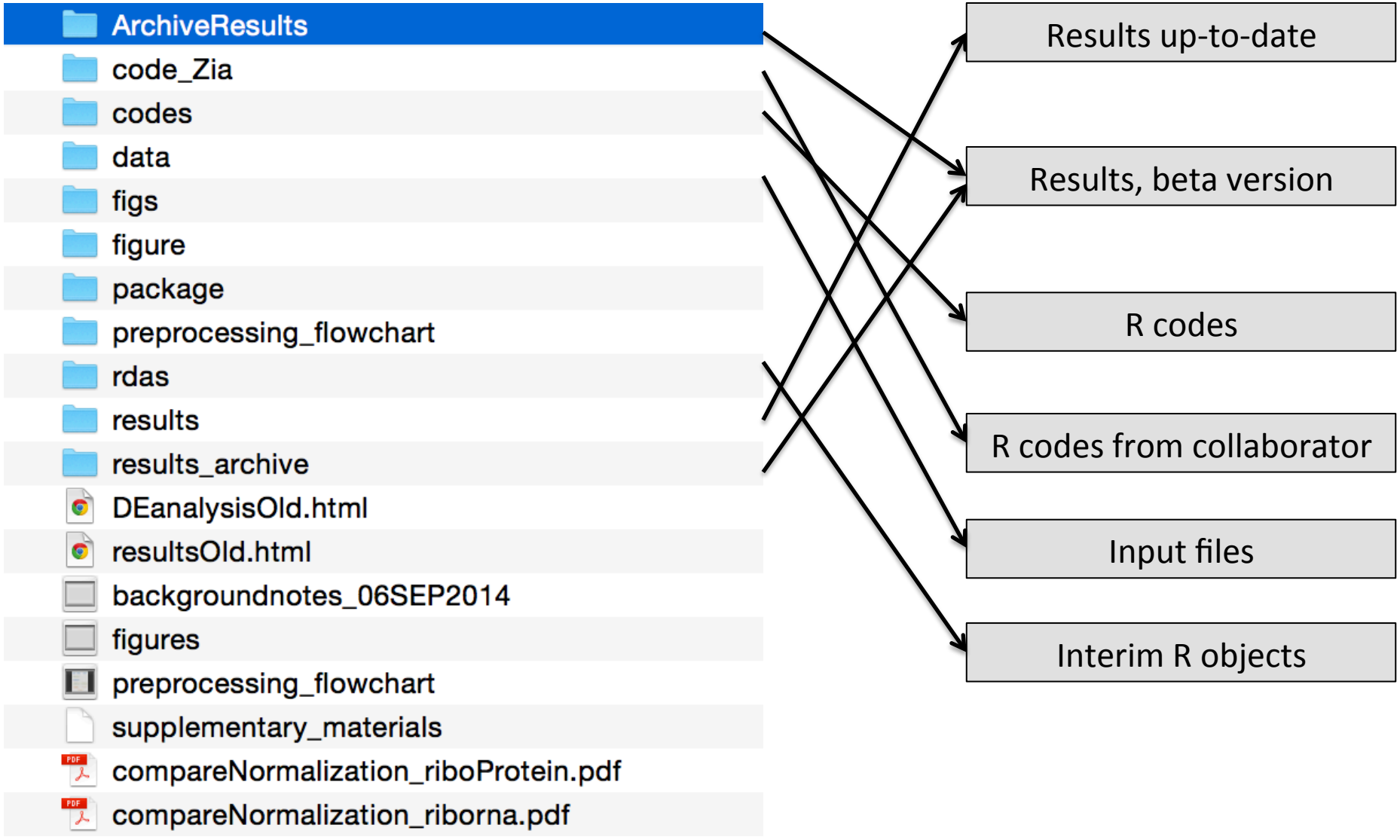
Joyce Hsiao

2015-12-03

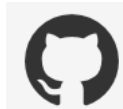
We work as a team



Before GitHub…



Recommended layout



- Analysis (ALL Rmarkdown files and htmls)
  - Figure (exported directly by user or as a by-product of Rmarkdown files)
  - Output (R objects, work-in-progress analysis results)
- Code (bash scripts, fresh R functions, or R functions that may not contribute to method development)
- Data (data that do not change frequently over the course the analysis, such as read count or phenotype information)
- Docs (method drafts, slides, etc.)

A workflow that works!

John Blischak says:

1. "Everything" goes into GitHub repo
2. Create a project website hosted on GitHub

https://github.com/jdblischak/singleCellSeq

Brief guide for first-timers

A. Making a project directory
B. Making htmls
C. Producing the website
D. Publishing the website
E. Add new analysis

Where I learned about this:
http://ialsa.github.io/tutorials/gh-pages-setup.html

A. Making a project directory

1. Clone ashlar and rename the repository

```
git clone https://github.com/jhsiao999/ashlar.git ashlar-trial
```

2. Open ashlar.Rproj (R project object) in the analysis directory. Once you do so, working directory of the current R session becomes *ashlar-trial/analysis*. No more specifying user-specific home directory. This is especially when working on collaborative projects.

3. Add your project information:
   - *analysis/*About.Rmd (project description)
   - *analysis/*Index.Rmd (homepage for the website)
   - *analysis/*License.Rmd (my default is Creative Common)
   - *analysis/*Template.Rmd (an example Rmarkdown template)
   - *ashlar/*README.md (github repo README)
   - *ashlar/*analysis/include/before_body.html (webpage header)

A. Making a project directory

4. Look at your .gitignore

   By default, all image files (png, pdf, etc.) and htmls are ignored in git add. We chose this to keep the master branch consist of only analysis files. Under this setup, you must force add (git add –f) in the first commit to the master branch and "every commit" to the gh-pages branch.

B. Making htmls

1. Run make command, which makes all changed Rmds.

```
cd ashlar-trial/anaysis
make
```

To force make htmls for all of the Rmds,

```
cd ashlar-trial/anaysis
make -B
```

2. Use knitr to compile Rmds into html files. If you work with RStudio, simply click on "knit html" in the tool bar.

B. Setting up GitHub repo

1. Reset git remote directory.

```
git remote rm origin
git remote add origin https:/github.com/jhsiao999/ashlar-trial.git
```

2. Go to to github.com. Create a repo called ashlar-trial. Then, commit all files

```
git add --all
git commit –m “first commit”
git push origin master
```

C. Producing the website (not publishing)

Approach 1:

Index.html is the table of content of your repository and contains hyperlinks to the listed content. You can use it as a TOC for your folder without ever publishing it. When published, index.html is the homepage of your webiste.

Approach 2: Use Jekyll

1. Setting up Jekyll pipeline. Here I assume you already have Ruby.

```
cd analysis
make
bundle exec jekyll serve
Localhost:4000
```

2. Producing the website

```
cd analysis
sudo gem install bundler
bundle install
```

D. Publishing website

Deploy to gh-pages.

```
git checkout gh-pages
git add –f .
git commit –m “build site”
git push origin gh-pages
```

Give it a minute, then wala!

https://jhsiao999.github.io/ashlar-trial

E. Add new analysis

If you've been working in the gh-pages branch,

```
git branch
cd analysis
git add –f *html figures/*
git commit –m "add new analysis"
git push origin gh-pages
```

Push the master branch to the gh-branches, run Make file and then push the updates to the gh-branches:

```
git checkout gh-pages
git merge master
cd analysis
make
git add --all
git commit –m "build site"
git push origin gh-pages
```

E. Add new analysis

If you've been working in the gh-pages branch,

```
git branch
cd analysis
git add –f *html figures/*
git commit –m "add new analysis"
git push origin gh-pages
```

Then to update the master branches with the changes, do

```
git checkout master
git merge gh-pages
git add *Rmd
git commit –m "add new analysis"
git push origin master
```

Likewise, if you work at the master branch, update the gh-pages branch when finished.

A useful Git tip

If > 1 person contributes to the repo on a regular basis, do the following to avoid disasters!

```
git checkout work-branch  ## move the pointer to local work branch
git add new_edits
git commit -m "new_edits"
git push origin work-branch  ## push edits to remove work branch

### At this point, you can make a merge and a pull request to review the edits

git checkout master  ## move the pointer to local master
git pull origin master  ## fetch and merge remote master to local master
git merge work-branch  ## merge local work-branch into master and update master
git checkout work-branch  ## move to local work-branch
git merge master  ## merge remote master to local work-branch
git push origin work-branch  ## move the pointer back to local work-branch

### At this point, the commit numbers of your work-branch and master should be the same!!!!!!!!!!!!!!!!!!!
```

https://github.com/jhsiao999/singleCell-method#collaborating-on-the-project

Other tips

You don't have write permissions for the /Library/Ruby/Gems/2.0.0 directory.

sudo gem update --system