

Informatics team manual of procedures

Ania Tassinari

2019-06-25

Contents

1	About	5
2	Best practices	7
2.1	Reproducible research	7
2.2	Version control	8
2.3	Code guidelines	8
3	Project workflows	9
3.1	Using Workflowr	9
3.2	Set Up Workflow and Executing	19
4	Toolbox	25
4.1	Command Line Tools	25
4.2	R	28
4.3	python	29
4.4	perl	29

Chapter 1

About

This is a manual of operations for the Agios Informatics team. Its purpose is to:

- provide a resource on **best practices** (Chapter 2)
- aid in setting up effective and reproducible **project workflows** (Chapter 3)
- promote learning and sharing of **ideas** (Chapter 4).

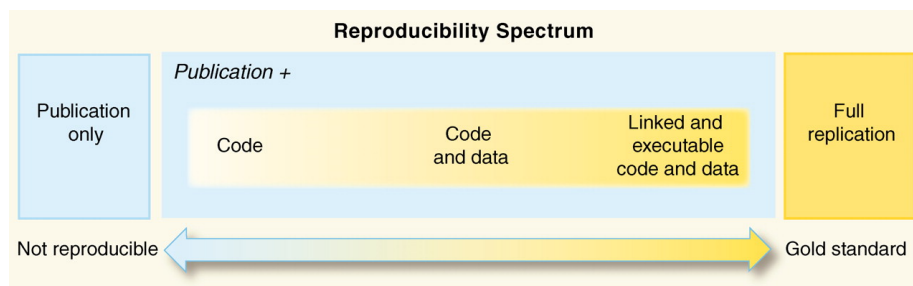
Chapter 2

Best practices

2.1 Reproducible research

2.1.1 Purpose

1. Improve collaborative analyses:
 - make sharing easier
 - enable retrieval and interpretation of results long after analysis ended
2. Simplify hand-off to Biostats
3. Improve confidence in our data and results



Source: Peng et al., *Reproducible Research in Computational Science*. Science 2011.

2.1.2 DO's and DON'T's of reproducible research

DO start with good science DON'T do things by hand Was any part of this analysis done by hand? - If so, are those parts precisely documented? - Does the documentation match reality? DON'T point and click DO teach a computer DO use version control DO keep track of your software environment DON'T save any output (until it's time to write a paper) DO set your seed

Source: Reproducible Research at Coursera

2.2 Version control

2.3 Code guidelines

2.3.1 R

- Tidyverse Style Guide
- Google's R Style Guide

Chapter 3

Project workflows

3.1 Using Workflowr

3.1.1 Quick Start

This is a quick version of everything listed below, if you want more clear or specific instructions then please skip this chapter and follow the steps in the following one.

3.1.1.1 Set Up

In the Console tab of RStudio in (None) project:

```
install.packages("workflowr")
library("workflowr")
wflow_git_config(user.name = "First Last", user.email = "first.last@agios.com")
```

3.1.1.2 Creating Projects

In the Console tab,

```
wflow_start("PROJECT_NAME")
wflow_build()
wflow_publish(c("analysis/*.Rmd"), "Publish the initial files for PROJECT_NAME")
```

3.1.1.3 Connecting to GitLab

In the **Console** tab,

```
wflow_use_gitlab(username = "first.last", repository = "PROJECT_NAME", domain = "ceres
```

Go to GitLab and do the following:

- Create a project in GitLab should the same name as the project in RStudio
 - We called ours PROJECT_NAME
- Go back to GitLab and and scroll down to the push an existing Git repository option
 - Then, copy every thing in the box besides the cd line
- Paste what you just copied into the **Terminal** tab in RStudio
 - Make sure you are in PROJECT_NAME directory

3.1.1.4 Creating a New File

In the **Console** tab,

```
wflow_open("analysis/NEW_FILE.Rmd")
wflow_build()
wflow_publish(c("analysis/*.Rmd"), "Publish the file NEW_FILE")
```

In the **Terminal** tab,

```
git push
```

3.1.1.5 Publish to GitLab without Rebuilding Sites

1. Edit the Rmd file and save
2. Run one of the following commands (doesn't matter)
 - wflow_build()
 - It doesn't matter if we build other files, they won't be added to git unless we add them in the next step
 - wflow_build("file.rmd")
 - Knit the file
3. wflow_git_commit("file.rmd", "This is your commit message")
4. Flip into the terminal and run git push()

3.1.2 Installation

3.1.2.1 Programs Needed

We are assuming that you already have RStudio and GitLab, for this implementation we are using the RStudio on the new sever which is version 1.2.1335.1.

If you don't have GitLab you need to have an account setup through Agios, if you don't have the updated RStudio you need to get access to the new server and then use the following link : hpc.agios.local

3.1.2.2 Installing Workflowr

1. Open RStudio and change proejct in the top right corner to (None)
 - Make sure you are in your home directory on RStudio as well, thus in the bottom right corner of your screen under **New Folder**, it is labled **Home** with a small house.
2. In the **Console** tab located in the bottom lefthand corner :

```
install.packages("workflowr")
```

3. Confirm you have access to Workflowr, in the **Console** tab:

```
library("workflowr")
```

3.1.2.3 Configure Git

*This only needs to be done once per laptop

In the **Console** tab:

```
wflow_git_config(user.name = "First Last", user.email = "first.last@agios.com")
```

3.1.3 Create Project

3.1.3.1 Start Project

In the **Console** tab:

```
wflow_start("PROJECT_NAME")
```

1. What does `wflow_start` do?
 - Creates a directory that contains all starting files
 - Changes your current directory to `PROJECT_NAME`
 - Starts a Git repo which we will connect to GitLab repository
2. What is the `analysis` folder for?
 - Contains all source R Markdown files (Rmd)
 - Includes: `index.rmd`
 - * Contains no R code but generates `index.html` which eventually runs the entire project
 - Contains `_site.yml`
 - Allows user to edit theme, navigation bar, menus ect.
 - Helpful link to customizing
3. What is the `docs` folder for?
 - Contains all HTML files for website
 - Note that this file will be empty until we build the project
 - Each HTML file is built from a corresponding Rmd file in the `analysis` folder
 - Contains any figures created by Rmd files
4. What about the data, code and output files?
 - These files are there for your use and thus can be deleted if desired

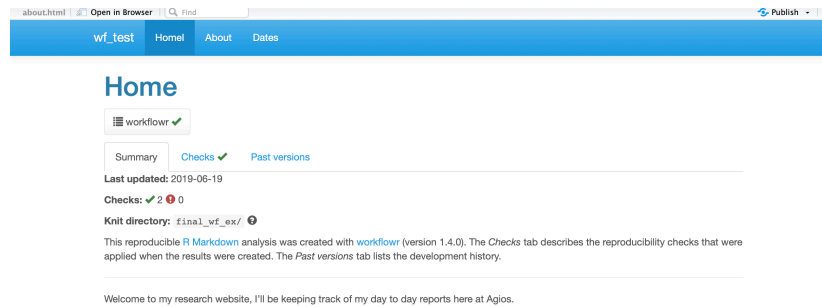
3.1.3.2 Build Project

In the `Console` tab:

```
wflow_build()
```

1. What does `wflow_build()` do?
 - Builds all the R Markdown files in `analysis` and saves their HTML in `docs`
 - Displays the website

Your website should look similar to the image of mine shown below (except with a `Publish` tab instead of a `Dates` tab)



3.1.3.3 View Project

At any time you can view the current site on your local machine by typing in the **Console** tab:

```
wflow_view()
```

3.1.3.4 Publish Website

Currently our project is simply an HTML file stored on our laptop, publishing the website will make it available online.

In the **Console** tab:

```
wflow_status()
```

This allows you to view which files are published or unpublished currently.

Now we want to publish our page the command to do so takes three parts

1. `c` - Commit
2. (“analysis/index.Rmd”, “analysis/about.Rmd”, “analysis/license.Rmd”),
- A character vector of the Rmd files you want published
 - It may be easier to place (“*.Rmd”) here to use all the files
3. “Publish the initial files” - A commit message to be posted

Overall, `wflow_publish` is a quick and error-free way for us to commit and push all of our Rmd files to GitLab at once.

In the **Console** tab:

```
wflow_publish(c("analysis/index.Rmd", "analysis/about.Rmd", "analysis/license.Rmd"), "1")
```

3.1.4 Connecting to GitLab

3.1.4.1 Creating a remote repository on GitLab

1. Login to GitLab and click **New Project**
2. The project name in GitLab should be the same name as the project in RStudio, we called ours `PROJECT_NAME`
3. Make sure to save it as **Internal** so everyone in Agios can see it

3.1.4.2 Connect RStudio and GitLab

1. Go to RStudio, in **Console** tab:

```
wflow_use_gitlab(username = "first.last", repository = "PROJECT_NAME", domain = "ceres")
```

2. Go back to GitLab and scroll down to the **push an existing Git repository** option
 - Then, copy every thing in the box besides the `cd` line

```
git remote rename origin old-origin
git remote add origin git@ceres.agios.com:Caitlin.Guccione/test-.git
git push -u origin --all
git push -u origin --tags
```

3. Go back into RStudio and in the **Terminal** tab
 - Make sure you are in the `PROJECT_NAME` repo
 - Paste the above commands
4. Return to GitLab to ensure your entire project exists there

3.1.5 Adding New Files

3.1.5.1 Creating New Files

Make sure you are inside the `PROJECT_NAME` project inside RStudio

In **Console** tab type:

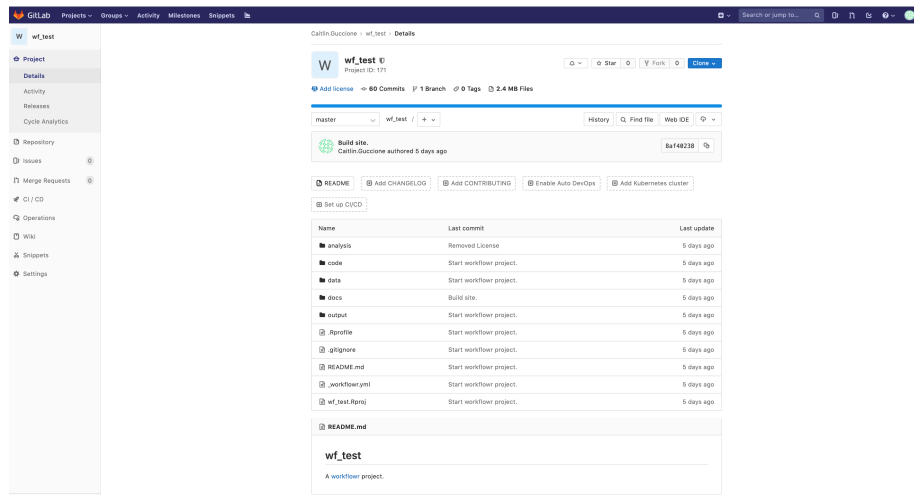


Figure 3.1: Example GitLab and Workflowr Connection

```
wflow_open("analysis/NEW_FILE.Rmd")
```

- This command creates a new Rmd file and then opens it for your convenience.

If we now want to see the HTML version of our file then we have two options:

1. In Console tab type:

```
wflow_build()
```

- You can add specific files to this command or simply leave it empty
- This produces a small view of your website right on RStudio

2. Press the 'Knit' button in RStudio as shown below:



- This produces a large web version of your current HTML file

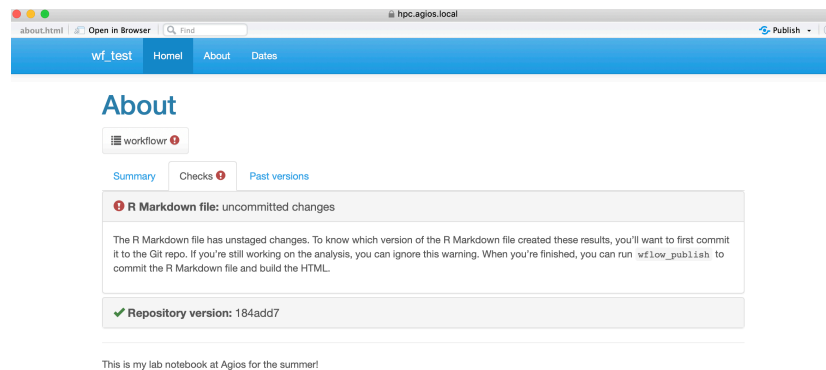
These steps will simply change the HTML file local but in order to make this public and add it to GitLab we need to update our changes.

3.1.5.2 Update your Changes

1. Check the status to see what needs to be updated, in the **Console** tab,

```
wflow_status()
```

This can also be done by looking at the red checks on the workflow section of your live page as shown below:



2. Make the appropriate HTML files public and updated, in the **Console** tab,

```
wflow_publish(c("analysis/index.Rmd", "analysis/NEW_FILE.Rmd"), "Add my first file")
```

- This is the same format found on the **Publish Website** tab of this page and so you can customize it in the same way

There is one exception to this and it's when you want to make updates to the `_site.yml` file found in the `analysis` folder. This file controls the style on the top of every page of your website. In this case you want to update all HTML files even though their Rmd files aren't changed.

In that case, use this,

```
wflow_publish("analysis/_site.yml", "Change the theme", republish = TRUE)
```

3. Push the final changes to GitLab

As we did previously in the **Publish Website**, in the **Terminal** tab,


```
git push
```

3.1.5.3 Adding Workflowr to New File

If you want the normal workflowr setup which is found on all the other pages, then replace the `---` part of the file with the following code:

```
---
title: "Home"
site: workflowr::wflow_site
output:
  workflowr::wflow_html:
    toc: false
editor_options:
  chunk_output_type: console
---
```

3.1.6 Styling the Webpage

3.1.6.1 Helpful Links

If you already have an idea of what you would like to change, below are a few very helpful resources filled with information:

- This resource is a great place to start because it has all basics of Rmd syntax and I used it as a cheat sheet along the way.
 - Rmd Cheat Sheet
- This is an entire book all about Rmd and how to use it. I found it rather lengthy but very helpful.
 - Rmd Thorough Guide
- If something isn't quite working right you may have run into a workflowr issue in which case their FAQ's page is helpful.
 - Workflowr FAQ's

3.1.6.2 Changing the Theme

Changing the theme modifies the overall appearance of the webpage is a quick and easy way to spice up the page.

1. Go into your `analysis/_site.yml` file

2. Underneath output add `theme = cerulean` as shown below:

- The cerulean theme matches Agios colors

```
output:
  theme: cerulean
```

3. Choose your theme

- The following themes are available : “default”, “cerulean”, “journal”, “flatly”, “darkly”, “readable”, “spacelab”, “united”, “cosmo”, “lumen”, “paper”, “sandstone”, “simplex”, “yeti”
- You can view how they look here: [Themes](#)

4. Preview your theme using,

```
wflow_build()
```

5. Update your website by running,

- This will rebuild every HTML file even if thier corresponding Rmd file hasn't been updated

```
wflow_publish("analysis/_site.yml", "Change the theme", republish = TRUE)
```

The following website will also walk you through changing the theme: [Themes Overview](#)

3.1.6.3 Adding Photos

Although this may seem like a simple task, it is a bit challenging since we are using Workflowr

1. Create a `photos` folder inside the `docs` folder and add your photo there:

```
dir.create("docs/photos")
```

2. Include the following command wherever you want your graphic to appear:

```
```{r a3, out.width='100%',fig.align='center',
 fig.cap='Example GitLab and Workflowr Connection',
 echo=FALSE}
knitr::include_graphics('photos/screen_shot.png')
```
```

3. View the images on the webpage

```
wflow_build()
```

4. Add to GitLab

- We need to push the actual photo to GitLab using `wflow_git_commit` and then we can use `wflow_publish` to automatically push the rest of the files to GitLab

```
wflow_git_commit("docs/assets/external.png", "Add external image of ...")  
wflow_publish()
```

3.2 Set Up Workflow and Executing

3.2.1 Create a folder for your *newproject*

Come up with a project structure you like and stick with it.

3.2.1.1 Copy from a previously created template folder

Use `cp -r project_template newproject`, where `project_template` has structure:

```
ania.tassinari at ubuntu07 in ~
[tree project_template/
project_template/
├── data
│   ├── interim
│   │   └── pheno
│   ├── processed
│   │   └── pheno
│   └── raw
│       ├── MAKE_EACH_FILE_READ_ONLY
│       └── pheno
├── README
├── results
│   ├── figures
│   ├── notebooks
│   │   └── notebook.Rmd
│   └── reports
├── src
└── sub
    ├── logs
    └── qsub

15 directories, 3 files
```

3.2.1.2 Use a bash script

Call `./setup_project.sh newproject`, where `setup_project.sh` is:

Don't forget! - Fill project README - Adapt structure to project needs -
Exclude data and other large files from git using `.gitignore` (see next section)
- Make files in `data/raw` read-only with `chmod -w`

Project organization ideas: http://projecttemplate.net/getting_started.html
Packaging data analytical work reproducibly using R (and friends) R workflow
fun Cookiecutter Data Science

3.2.2 Set up a repository for your code on Agios' secure GitLab

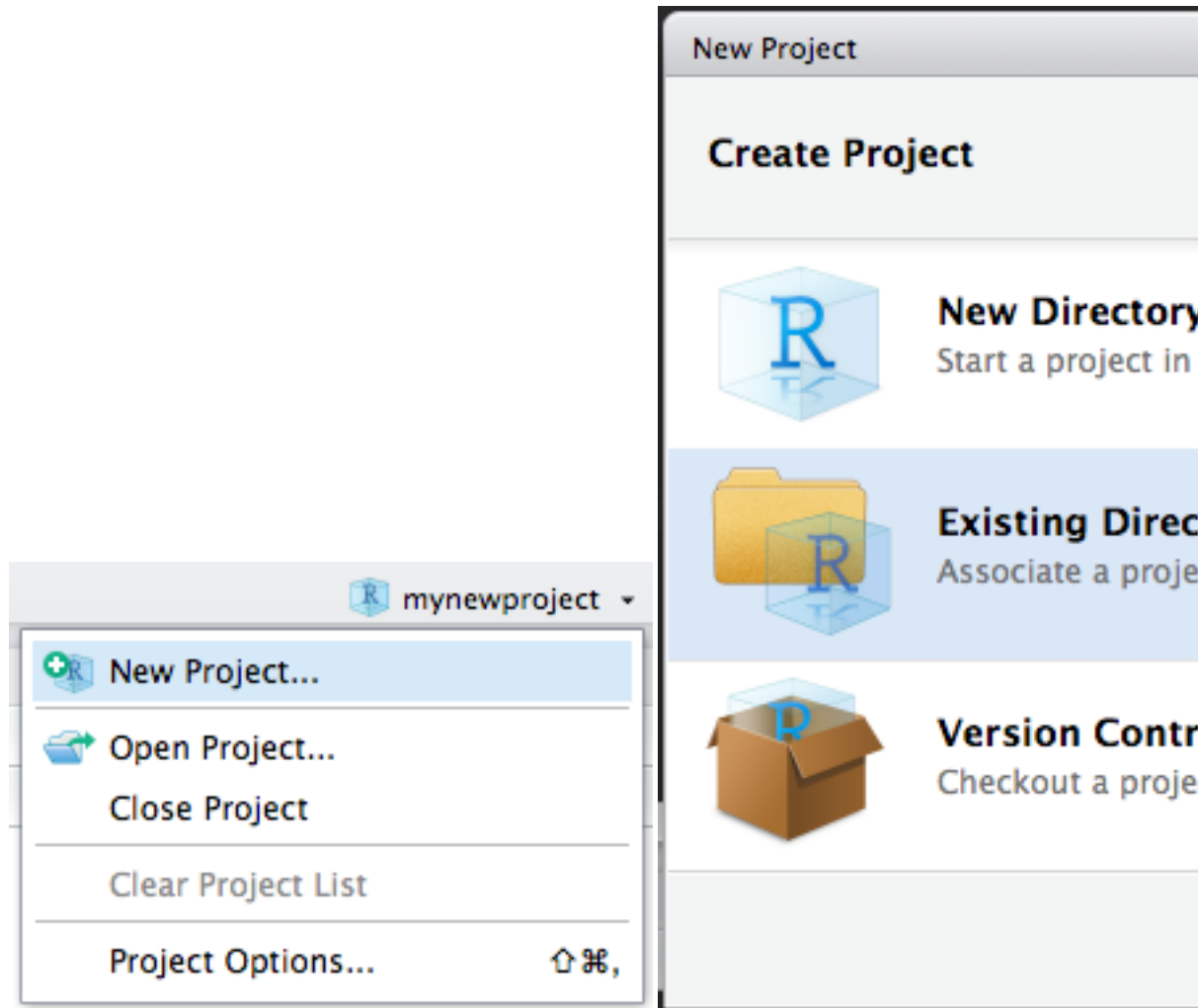
Create a new project at <http://ceres.agios.com> (Mark P. can help)

3.2.3 Set up a repository for your code locally and link to GitLab

In your *newproject* folder on command line execute (modify user name):

3.2.4 Set up an R project in RStudio

Choose Existing Directory (*newproject*)



3.2.5 Analysis in R and RStudio

Data:

- Raw data:
 - If accessed from the web, include url, description, and date accessed in README
- Processed:
 - Processed data should be named so it is easy to see which script generated the data

- * Can add file descriptions to `filename.README` and place processing script in the same directory as data (works well for preprocessing steps, like alignments, etc)
- Processed data should be tidy

Code:

- Place (almost) all intermediate scripts in `newproject/src/`
- Any chunks of code frequently reused in the analysis should be converted into functions, saved in `newproject/src/functions.R`, and sourced in scripts, notebooks and reports.
- Use Google's R Style Guide or The tidyverse styleguide to format your code and make it easier to read (if need be run code through `formatR`)

Figures:

- Exploratory:
 - Don't have to be pretty
 - Can be embedded in report / notebook
- Final:
 - Should be polished and saved in `newproject/results/figures/`

Scripts:

- Raw:
 - May be less commented (but comments help you!)
 - May be multiple versions
 - May include analyses that are later discarded
- Final:
 - Clearly commented
 - Small comments liberally - what, when, why, how
 - Bigger commented blocks for whole sections
 - Include processing details
 - Only analyses that appear in the final write-up

Notebooks and reports:

- R markdown files can be used to generate reproducible reports
- Text and R code are integrated

- Notebooks:
 - intermediate
 - may use one per day or one per subanalysis
 - documents all attempts
- Reports:
 - final methods and results only
 - good for sharing

Adapted from: Reproducible Research at Coursera

3.2.6 Version control in git and GitLab

Adopt a branching workflow appropriate for the project and team size, and stick to it.

gitforsmallteams

Reprinted from: Git workflow for small teams. Link currently is password protected.

git and git-workflow resources: Learn git Git branching model GitFlow

3.2.7 Keeping track of enviroment

Use `devtools::session_info()`

or `sessionInfo()`

or `docker` with `rrtools`

Chapter 4

Toolbox

4.1 Command Line Tools

4.1.1 Bioinformatics

4.1.1.1 Get average read length from a .bam file

```
samtools view sorted.bam | head -n 1000000 | cut -f 10 | perl -ne 'chomp;print length($_) . "\n"
```

4.1.2 Docker

4.1.2.1 Basic commands

Here are a few helpful commands for working with docker images

```
docker ps -a          # Lists containers (and tells you which images they are spun from)
docker images         # Lists images
docker rm <container_id> # Removes a container

docker rmi <image_id>  # Removes an image
                     # Will fail if there is a running instance of that image i.e. container

docker rmi -f <image_id> # Forces removal of image even if it is referenced in multiple repositories
                     # i.e. same image id given multiple names/tags
                     # Will still fail if there is a docker container referencing image
```

4.1.2.2 Pruning

```
# remove dangling images
docker rmi $(docker images --filter "dangling=true" -q --no-trunc)

# find other untagged images (with possible children) (<none>:<none>)
docker images -a | grep "none" | awk '{print $3}'

# try removing them
docker rmi $(docker images -a | grep "none" | awk '{print $3}')
```

*# if any left because of clingy children, get __parent.ID__ (ID of untagged image) and
example: docker inspect --format='{{.Id}} {{.Parent}}' \$(docker images --filter since=*
docker inspect --format='{{.Id}} {{.Parent}}' \$(docker images --filter since=__parent.

*# then remove listed __children.id__ one by one
example: docker rmi 382096f13260254f3c472bf63f063b8ecbc2d4cc06fe7a940d6fbd4636ef77b1*
 docker rmi __child.id__

4.1.3 File system

4.1.3.1 List top 5 largest files

```
du -a /path/to/my/dir/ | sort -n -r | head -n 5
```

Example:

```
du -a /bin | sort -n -r | head -n 5
```

```
## 17652    /bin
## 2016    /bin/busybox
## 1552    /bin/ksh93
## 1088    /bin/bash
## 816     /bin/zsh
```

4.1.3.2 List files in a folder separated by delimiter

```
ls -l /path/to/my/dir/ | paste -sd "," -
```

Example:

```
ls -l /bin | paste -sd "," -
```

```
## bash,btrfs,btrfsck,btrfs-debug-tree,btrfs-find-root,btrfs-image,btrfs-map-logical,btrfs-select
```

4.1.3.3 Compare structure of two directories

```
vimdiff <(cd dir1; find . | sort) <(cd dir2; find . | sort)
```

Example:

```
./07272018_AG120_Heme
./07272018_AG120_Heme/RNA_64103533178_tumor_rna
./07272018_AG120_Heme/RNA_64103533178_tumor_rna/gene_abundances.tsv
./07272018_AG120_Heme/RNA_64103533178_tumor_rna/input_fastq_source.txt
./07272018_AG120_Heme/RNA_64103638149_tumor_rna
./07272018_AG120_Heme/RNA_64103638149_tumor_rna/gene_abundances.tsv
./07272018_AG120_Heme/RNA_64103638149_tumor_rna/input_fastq_source.txt
./07272018_AG120_Heme/RNA_64103757263_tumor_rna
./07272018_AG120_Heme/RNA_64103757263_tumor_rna/gene_abundances.tsv
./07272018_AG120_Heme/RNA_64103757263_tumor_rna/input_fastq_source.txt
./07272018_AG120_Heme/RNA_64103757270_tumor_rna
./07272018_AG120_Heme/RNA_64103757270_tumor_rna/gene_abundances.tsv
./07272018_AG120_Heme/RNA_64103757270_tumor_rna/input_fastq_source.txt
./07272018_AG120_Heme/RNA_64103757273_tumor_rna
./07272018_AG120_Heme/RNA_64103757273_tumor_rna/gene_abundances.tsv
./07272018_AG120_Heme/RNA_64103757273_tumor_rna/input_fastq_source.txt
./07272018_AG120_Heme/RNA_64103800434_tumor_rna
./07272018_AG120_Heme/RNA_64103800434_tumor_rna/gene_abundances.tsv
./07272018_AG120_Heme/RNA_64103800434_tumor_rna/input_fastq_source.txt
./07272018_AG120_Heme/RNA_64103811058_tumor_rna
./07272018_AG120_Heme/RNA_64103811058_tumor_rna/gene_abundances.tsv
./07272018_AG120_Heme/RNA_64103811058_tumor_rna/input_fastq_source.txt
./07272018_AG120_Heme/RNA_64103968525_tumor_rna
./07272018_AG120_Heme/RNA_64103968525_tumor_rna/gene_abundances.tsv
./07272018_AG120_Heme/RNA_64103968525_tumor_rna/input_fastq_source.txt
```

4.1.4 Sun Grid Engine

4.1.4.1 Deprioritize jobs queued on SGE

```
qalter -p -100 {jobid1..jobidn}
```

4.1.5 vim

4.1.5.1 Repeat content of line in new column

```
# repeat content of line  
# a  
# b  
# c  
# becomes  
# a = C.a  
# b = C.b  
# c = C.c  
:%s/./& = C.&
```

4.2 R

4.2.1 Heatmaps

4.2.2 Addition

```
x <- 3  
y <- 4  
z <- x+y  
z
```

```
## [1] 7
```

4.3 python

4.4 perl

4.4.0.1 Get number of lines in a file

```
open(my $input, "-|", "wc -l < $fastqs");  
my $rc = <$input>;  
if ($rc =~ /\d+/) {  
    print $rc;  
}
```