Big jobs/simulations
Tools for Reproducible Research

Karl Broman
Biostatistics & Medical Informatics, UW–Madison

kbroman.org
github.com/kbroman
@kwbroman
Course web: kbroman.org/Tools4RR
Suppose I've just written an R function and it seems to work, and suppose I noticed a simple way to speed it up.

What should I do first?
But first…

Suppose I've just written an R function and it seems to work, and suppose I noticed a simple way to speed it up.

What should I do first?

► Make it an R package
But first…

Suppose I've just written an R function and it seems to work, and suppose I noticed a simple way to speed it up.

What should I do first?

- Make it an R package
- Write a test or two
Suppose I've just written an R function and it seems to work, and suppose I noticed a simple way to speed it up.

What should I do first?

- Make it an R package
- Write a test or two
- Commit it to a git repository
So what's the big deal?

- You don't want `knitr` running for a year.
- You don't want to re-run things if you don't have to.
Unix basics

nice +19 R CMD BATCH input.R output.txt &
fg
ctrl-Z
bg
ps ux
top
kill
kill -9
pkill
Disk thrashing

In computer science, thrashing occurs when a computer's virtual memory subsystem is in a constant state of paging.

– Wikipedia
Disk thrashing

In computer science, thrashing occurs when a computer's virtual memory subsystem is in a constant state of paging, rapidly exchanging data in memory for data on disk, to the exclusion of most application-level processing.

– Wikipedia
Biggish jobs in knitr

- Manual caching
- Built-in `cache=TRUE`
- Split the work and write a Makefile
```{r a_code_chunk}
file <- "cache/myfile.RData"

if(file.exists(file)) {
    load(file)
} else{

    ....

    save(object1, object2, object3, file=file)
}
```
Chunk references

```r
``` not_shown, eval=FALSE

code_here <- 0
```n

```
``` a_code_chunk, echo=FALSE

file <- "cache/myfile.RData"

if(file.exists(file)) {
  load(file)
} else{
  ``` not_shown ```
  save(code_here, file=file)
}
```n
```
A cache gone bad
Knitr's cache system

```r
```r chunk_name, cache=TRUE}
load("a_big_file.RData")
med <- apply(object, 2, median, na.rm=TRUE)
```r

- Chunk is re-run if edited.
- Otherwise, objects from previous run are loaded.
- Don't cache things with side effects
  
  e.g., `options()`, `par()`
Cache dependencies

Manual dependencies

```r
```{r chunkA, cache=TRUE}
Sys.sleep(2)
x <- 5
```

```r
```{r chunkB, cache=TRUE, dependson="chunkA"}
Sys.sleep(2)
y <- x + 1
```

```r
```{r chunkC, cache=TRUE, dependson="chunkB"}
Sys.sleep(2)
z <- y + 1
```
Cache dependencies

Automatic dependencies

```r
```
Parallel computing

If your computer has multiple processors, use `library(parallel)` to make use of them.

- `detectCores()`
- `RNGkind("L'Ecuyer-CMRG")` and `mclapply` (Unix/Mac)
- `makeCluster`, `clustersetRNGStream`, `clusterApply`, and `stopCluster` (Windows)
Systems for distributed computing

- HTCondor and the UW-Madison CHTC
- Other condor-like systems
- "By hand"
  - e.g., perl script + template R script
Simulations

- Computer simulations require RNG seeds (.Random.seed in R).

- Multiple parallel jobs need different seeds.

- Don't rely on the current seed, or on having it generated from the clock.

- Use something like `set.seed(91820205 + i)`

- An alternative is create a big batch of simulated data sets in advance.
Save everything

- RNG seeds
- input
- output
- version numbers, with `sessionInfo()`
- raw results
- script to combine results
- combined results
- `ReadMe` describing the point
One Makefile to rule them all

- Separate directory for each batch of big computations.
- Makefile that controls the combination of the results (and everything else).
- KnitR-based documents for the analysis/use of those results.
Potential problems

- Forgetting `save()` in your distributed jobs
- A bug in the `save()` command
- `make` clobbers some important results
  - Scripts should refuse to overwrite output files
Summary

- Careful organization and modularization.
- Save everything.
- Document everything.
- Learn the basic skills for distributed computing.