Reproducibility with git and rmarkdown

Thomas J. Leeper

Department of Government
London School of Economics and Political Science

30 January 2018
Version Control as Organization

- Version control helps you stay organized
Version Control as Organization

- Version control helps you stay organized
  1. What’s important to keep around?
Version Control as Organization

- Version control helps you stay organized
  1. What’s important to keep around?
  2. What’s not important to keep around?
Version Control as Organization

- Version control helps you stay organized
  1. What’s important to keep around?
  2. What’s not important to keep around?
  3. What is all this crap?
Version Control as Organization

- Version control helps you stay organized
  1. What’s important to keep around?
  2. What’s not important to keep around?
  3. What is all this crap?

- Think “tracked changes” for all of your files
Version Control as Organization

- Version control helps you stay organized
  1. What’s important to keep around?
  2. What’s not important to keep around?
  3. What is all this crap?

- Think “tracked changes” for all of your files
  - Save history of changes/versions
Version Control as Organization

- Version control helps you stay organized
  1. What’s important to keep around?
  2. What’s not important to keep around?
  3. What is all this crap?

- Think “tracked changes” for all of your files
  - Save history of changes/versions
  - Experiment non-destructively
Version Control as Organization

- Version control helps you stay organized
  1. What’s important to keep around?
  2. What’s not important to keep around?
  3. What is all this crap?

- Think “tracked changes” for all of your files
  - Save history of changes/versions
  - Experiment non-destructively
  - Collaborate
Version Control as Organization

- Version control helps you stay organized
  1. What's important to keep around?
  2. What's not important to keep around?
  3. What is all this crap?

- Think “tracked changes” for all of your files
  - Save history of changes/versions
  - Experiment non-destructively
  - Collaborate

- You’re probably already version controlling informally!
<table>
<thead>
<tr>
<th>Background</th>
<th>Git</th>
<th>Intermediate Git</th>
<th>Branches &amp; Remotes</th>
</tr>
</thead>
</table>

![Image of three comic panels](image-url)

- **FINAL.doc!**
- **FINAL_rev.2.doc**
BACKGROUND

Git

Intermediate Git

Branches & Remotes

FINAL_rev.6.COMMENTS.doc

FINAL_rev.8.comments5.CORRECTIONS.doc
FINAL_rev.18.comments7.corrections9_MORE.30.doc
FINAL_rev.22.comments49.corrections10.#@$%WHYDIDICOMETOGRADSSCHOOL????.doc
<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissertation - Prospectus 0.doc</td>
<td>2015-07-15 09:17</td>
<td>Microsoft Word 9...</td>
<td>18 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 1.doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>142 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 2.doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>246 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 3.doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>250 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 4.doc</td>
<td>2015-07-15 09:19</td>
<td>Microsoft Word 9...</td>
<td>250 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 5.doc</td>
<td>2015-07-15 09:19</td>
<td>Microsoft Word 9...</td>
<td>263 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 6.doc</td>
<td>2015-07-15 09:19</td>
<td>Microsoft Word 9...</td>
<td>287 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 7.doc</td>
<td>2015-07-15 09:19</td>
<td>Microsoft Word 9...</td>
<td>291 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 8.doc</td>
<td>2015-07-15 09:19</td>
<td>Microsoft Word 9...</td>
<td>300 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 9 (For Jamie).pdf</td>
<td>2015-07-15 09:19</td>
<td>PDF File</td>
<td>328 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 9.doc</td>
<td>2015-07-15 09:19</td>
<td>Microsoft Word 9...</td>
<td>340 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 10 (Big Question).doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>19 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 10 (New).doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>179 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 11 (Outline).doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>287 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 12.doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>175 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 12a (Outline).doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>40 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 12b (Outline).doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>50 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 13.doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>159 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 13a (Outline).doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>228 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 13b (Outline).doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>266 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 13c.doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>320 KB</td>
</tr>
<tr>
<td>Dissertation - Prospectus 14 (Methods Draft for Jamie).doc</td>
<td>2015-07-15 09:18</td>
<td>Microsoft Word 9...</td>
<td>44 KB</td>
</tr>
</tbody>
</table>
Wait, but why do we care?

If we’re going to be transparent *in the end* (e.g., at replication or data archiving stage), what do we need to provide?
Wait, but why do we care?

If we’re going to be transparent *in the end* (e.g., at replication or data archiving stage), what do we need to provide?

- A well-organized, reproducible analysis!
Wait, but why do we care?

If we’re going to be transparent *in the end* (e.g., at replication or data archiving stage), what do we need to provide?

- A well-organized, reproducible analysis!

So rather than make that an annoying, post-hoc exercise related to publication, try to get organized and stay organized throughout your project from the very beginning.
"Reproducibility is collaboration with people you don't know, incl. yourself next week." – @philipbstark #openscience
Git

- Git is “an open-source distributed version control system”
- Developed in 2005 by Linus Torvalds
- Widely used in software development world
Why use Git for open science?

- Helps you keep and annotate snapshots of your project over time
- Better than renaming your files all the time
- Better than using within-file VCS (e.g., Word)
- Better than single-stream sharing (e.g., Dropbox)
- Facilitates collaboration (incl. with future you)
- It's FOSS with lots of clients, tools, and community support
- Widely used in software development world
Why use Git for open science?

- Helps you keep and *annotate* snapshots of your project over time
  - Better than renaming your files all the time
  - Better than using within-file VCS (e.g., Word)
  - Better than single-stream sharing (e.g., Dropbox)
Why use Git for open science?

- Helps you keep and *annotate* snapshots of your project over time
  - Better than renaming your files all the time
  - Better than using within-file VCS (e.g., Word)
  - Better than single-stream sharing (e.g., Dropbox)

- Facilitates collaboration (incl. with future you)
Why use Git for open science?

- Helps you keep and *annotate* snapshots of your project over time
  - Better than renaming your files all the time
  - Better than using within-file VCS (e.g., Word)
  - Better than single-stream sharing (e.g., Dropbox)

- Facilitates collaboration (incl. with future you)

- It’s FOSS with lots of clients, tools, and community support
  - Widely used in software development world
Using Git

- Git create a “local repository” file that you can interact with using a number of tools
  - Command-line git
  - Git Bash
  - Git GUI
  - GitHub Desktop
  - RStudio (via “Projects”)
  - GitHub/Bitbucket/GitLab web interfaces
  - Gitkraken
  - git2r (R package)
  - ...

---

Background Git Intermediate Git Branches & Remotes

---

LSE
Git Essentials

1. stage
2. commit
3. branch
4. merge
5. push and pull
Git Essentials

1. **stage**
   - **stage**: select files to be recorded in a “snapshot” of the project
   - **unstage**: remove files from the snapshot (but not from your computer)

2. **commit**

3. **branch**

4. **merge**

5. **push and pull**
Git Essentials

1. stage
2. commit
   - commit: record a permanent snapshot of the staged files, labelled with a “commit message”
   - amend: modify (typically the most recent) commit with new changes or commit message
3. branch
4. merge
5. push and pull
Git Essentials

1 stage
2 commit
3 branch
   - produce a complete local copy of the project where changes can be made independently of the “master” branch
4 merge
5 push and pull
Git Essentials

1. stage
2. commit
3. branch
4. merge
   - update a branch with changes from another local branch (or a remote); you can change multiple branches independently.
5. push and pull
Git Essentials

1. stage
2. commit
3. branch
4. merge
5. push and pull
   - **push**: send the project (any new commits) to a remote server (like GitHub)
   - **pull**: grab new commits from a remote server
Git Essentials

1. stage
2. commit
3. branch
4. merge
5. push and pull
Hands-on practice!
git --version

git

git config --global user.name "My Name"
git config --global user.email "me@example.com"
git config --list
```
git init

```
```
git status
```
```

```
`echo Hello world! > README.md`  
```
```
git add README.md
```
```
git status
```
```
```
`git rm --cached README.md`  
```
```
git status
```
```
```
git add --all
```
```
git commit -m "my first commit!"
```
```
git status
```
```
Initializing a Project Structure

- There’s no single best way to organize a project
- But, some words of wisdom:
  - Put like with like
  - Avoid excessive hierarchy
  - Not everything needs to go into git
  - Steal others’ structures!
What makes up the ideal reproducible research product?

- Gandrud’s template
- rOpenSci’s “Research Compendium”
- Project TIER
project
|-- DESCRIPTION   # project metadata and dependencies
|-- README.md     # top-level description of content

|-- data/
  |-- my_data.csv # raw data, not changed once created

|-- analysis/
  |-- my_scripts.R # R code used to analyse data
TIER Protocol Documentation

### Original Data
- Original data files
- Importable data files (if necessary)

### Metadata
- The Metadata Guide
- Supplementary metadata documents (if necessary)

### Analysis Data
- Analysis data files

### Documents
- The final paper
- The Data Appendix
- The Read Me file

### Command Files
- Command files
mkdir code
mkdir data
mkdir figures
git status
git status

`cat README.md`

`# do something to README.md`

git diff

git add README.md

git commit -m "second commit"

git status

git log

git log --oneline

git log --oneline -1

git log --oneline --stat
git status

```bash
git diff README.md  
git diff HEAD README.md
```
!! DANGER: Amend Commit !!

git status

```bash
git log --oneline
```

# maybe add/rm files

```bash
git amend
```

# enter the hell of vim

```bash
git config --global core.editor "<executable> <options>"
```
Safe reversion

git status

git log --oneline

git revert <commit hash>

# enter the hell of vim

# or something else terrible

git revert --abort
!! DANGER: Unsafe reversion !!

The StackOverflow Question
```bash
git status
echo "bad bad bad" > bad.txt
git status
echo bad.txt > .gitignore
git status
echo bad bad bad bad > bad1.txt
echo bad bad bad bad > bad2.txt
echo bad* > .gitignore
git status
git add bad1.txt -f
git status
```
Navigating History

git status

```bash
git log
```
git status

```
git log
```  
```
git checkout <commit hash>
git status
```
```
ls
```  
```
echo aaaaaaah!>manuscript.txt
```  
```
git checkout master
```
This is Git. It tracks collaborative work on projects through a beautiful distributed graph theory tree model.

Cool. How do we use it?

No idea. Just memorize these shell commands and type them to sync up. If you get errors, save your work elsewhere, delete the project, and download a fresh copy.
Branches

- Branches are local, parallel versions of your entire project

- Useful for multiple things:
  - Experimentation
  - Manuscript submissions
  - Collaboration
Source: https://www.atlassian.com/git/tutorials
Source: https://www.atlassian.com/git/tutorials
Simple branch and merge

```
# Set up

git status

git checkout -b thomas

git status

# do something

git add --all

git commit -m "thomas's commit"

git checkout master

git branch

git log --graph --oneline

git merge thomas
```
GUIs

- You can do everything in Git on the command line

- GUIs can be helpful for:
  - Exploring history
  - Visualizing branches
  - Confirming what you’re doing
Merge conflicts

git checkout -b thomas

# do something to README.md

git add --all

git commit -m "change on thomas"

# do something to README.md

git add --all

# do something to README.md

git log
Remotes

- A server ("cloud") instance of the Git repository

- Useful for multiple things:
  - Collaboration
  - Transparency
  - Archiving/backups
  - Using web-based Git interfaces
Remotes

- Three major players in cloud Git
  - GitHub
  - Atlassian Bitbucket
  - GitLab

- Why choose one or the other?
  - Cost
  - Collaborators
  - Private repositories
git status

```shell
git remote add github https://github.com/leeper/rt2
```

```shell
git remote

git remote set-url

git remote rename

git remote remove
```
git status
git push github master -u
git fetch github
git fetch github master
git checkout -b new-idea
git push github new-idea
git checkout master
git pull github master
git pull
git status

```bash
git tag -a v0.0.1 -m "v0.0.1"
git push --tags

```
git tag -d v0.0.1
Tags versus Branches

- **Branches** are for working versions of project
  - Collaborator-specific branches
  - Submission-specific branches
  - Experimental or “bug fix” branches

- **Tags** are for marking particular snapshots
  - Significant moments in project history
  - Journal submission or conference version
  - Formal “releases”
Collaboration

- **Technical aspects**
  - Give collaborators access on GitHub (or wherever)
  - Work on separate branches
  - Merge agreed changes into `master`

- **Human factors aspects**
  - Requires agreeing on workflow
  - Communication about what goes in “master”
  - Can feel awkward if moving from a Dropbox- or email-based collaboration style
Try it with a partner!

1. Partner A create a GitHub repo; give Partner B access
2. Partner B should `git fetch`/`git pull` the repo
3. Partner B should create a local branch and `git push`
4. Partner A should `git fetch` the branch
5. Partner A should `git merge` the branch to `master` and `git push`
6. Partner B should `git pull` from `master`
7. Both use `git log` to compare
Conclusion

- Once you use Git, you’ll never want to go back to your old workflow
**Conclusion**

- Once you use Git, you’ll never want to go back to your old workflow

- But, collaborators probably don’t know or want to use Git!
Conclusion

- Once you use Git, you’ll never want to go back to your old workflow

- But, collaborators probably don’t know or want to use Git!

- Git is crazy complicated — StackOverflow is your friend
How do you typically get figures, tables, and other material out of analytic software and into papers or presentations?
Dynamic Documents in R

- The dynamic documents landscape is evolving very, very rapidly:
  - Early 2000s: Sweave
  - 2010’s: knitr
  - Ongoing: Rmarkdown
Dynamic Documents in R

- The dynamic documents landscape is evolving very, very rapidly:
  - Early 2000s: Sweave
  - 2010’s: knitr
  - Ongoing: Rmarkdown

- Embed code (R or otherwise) inside a manuscript that outputs:
  - Word (.docx)
  - HTML
  - LaTeX/PDF
  - HTML or PPT slides
# My Manuscript
Thomas J. Leeper
This is my manuscript.
Rmarkdown

1. YAML metadata header
2. Document contents in markdown
3. Code in “code chunks”:

```
```{r chunk1}
# R code
hist(rnorm(1000))
```
```
This is my manuscript.

```r
hist(rnorm(1000))
```
Markdown Basics

Markdown is a very simple markup language for formatting simple texts:

*italics*
*bold*
‘preformatted‘
# Heading
## Heading
### Heading
[link](https://google.com)
Chunk Options

```r
{r chunk1, eval=TRUE, echo=TRUE}
2 + 2
```

```r
{r chunk2, eval=TRUE, echo=FALSE}
2 + 2
```

```r
{r chunk3, echo=FALSE, results="hide"}
2 + 2
```
Global Chunk Options

```
\{r options, eval = TRUE, echo = FALSE
library("knitr")
opts_chunk$set(echo = FALSE,
cache = TRUE,
message = FALSE)
```

Basic Tables

```
#{r table1, results = "asis"}
xtable::xtable(table(mtcars$cyl, mtcars$gear))

knitr::kable(head(mtcars))
```

Regression Results Tables

```
library("stargazer")
stargazer(
  x1 <- lm(mpg ~ disp + wt, 
            data = mtcars),
  x2 <- lm(mpg ~ disp + wt + vs, 
            data = mtcars),
  header = FALSE
)
```

Figures

```
fig1 = ggplot(mtcars, 
   aes(x = wt, 
       y = mpg, 
       colour = factor(cyl))) + 
   geom_point()
```

```
fig.cap = "Fuel Economy by Weight",
fig.height = 4,
fig.width = 6}
```