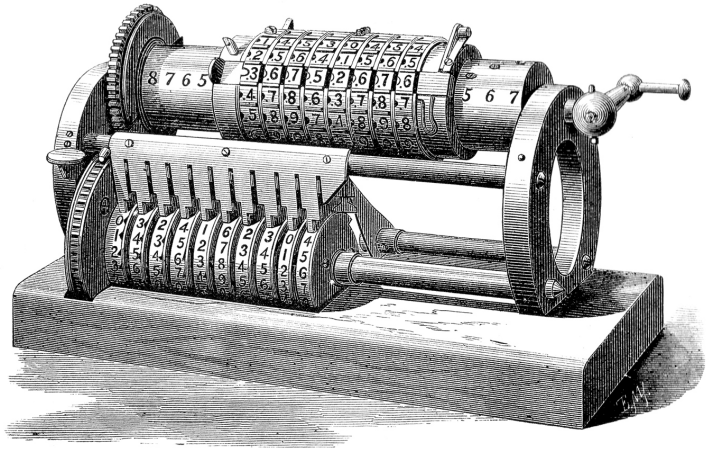
Programming in R

Binder, Bender,
Burk



Unit 1: Git

Git

Our course has multiple goals:

- Learn things about the R language: "R"
- Get to know nice tools to use: "Tools"
- Learn things about software development in general: "Dev"

This unit:

- "Tools" Track: git

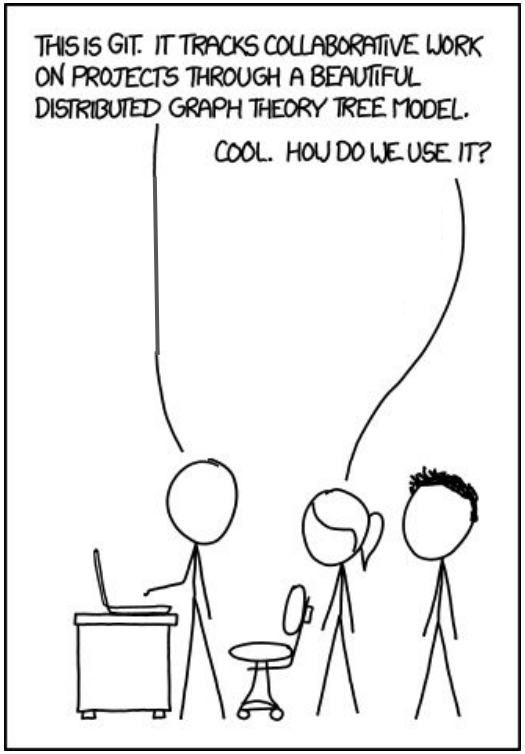
About this Unit

This is an introductory unit, meant to get you familiar with our style of instruction. We therefore only cover one topic, git, which you may even be familiar with already.

I think you should still scroll through these slides and see if they are helpful. The following weeks will cover more advanced topics.

Git

Git



https://xkcd.com/1597/

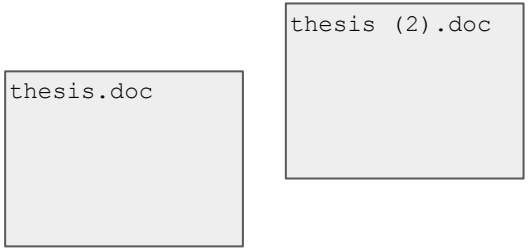
Git -- "Version Control System" (VCS)

You probably know this:

```
thesis.doc
```

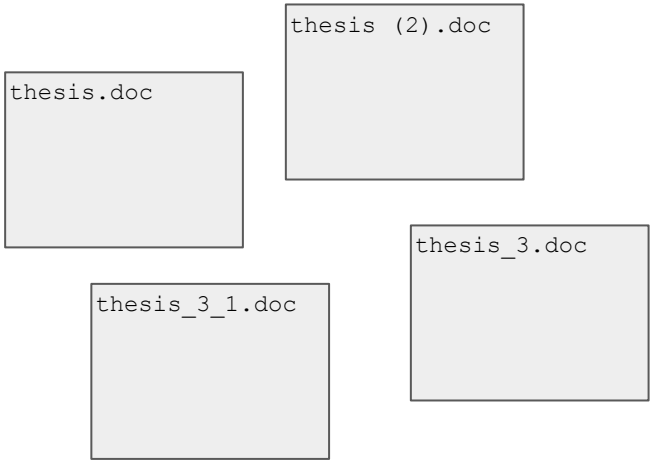
Git -- "Version Control System" (VCS)

You probably know this:



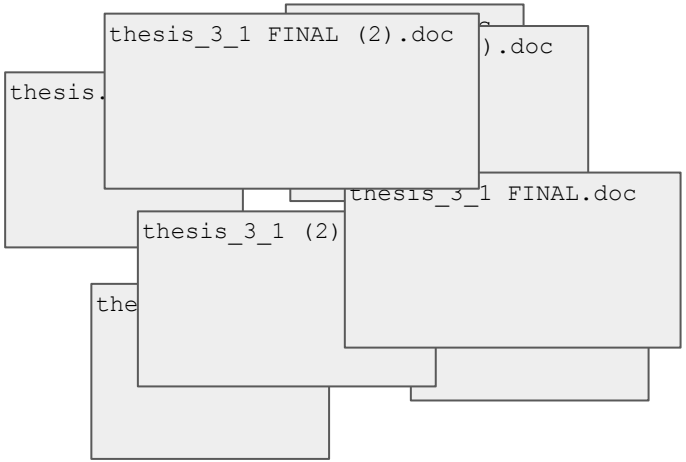
Git -- "Version Control System" (VCS)

You probably know this:



Git -- "Version Control System" (VCS)

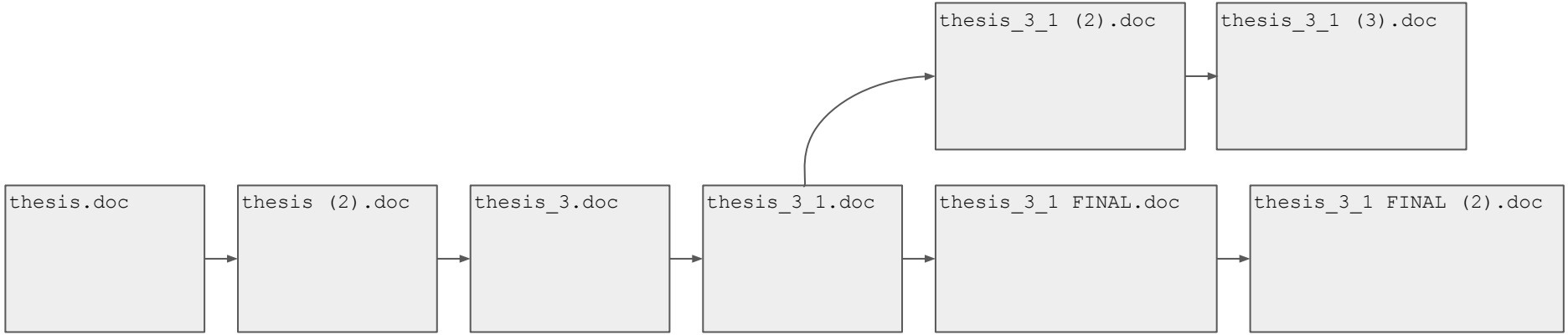
You probably know this:



Git -- "Version Control System" (VCS)

How about we organize things:

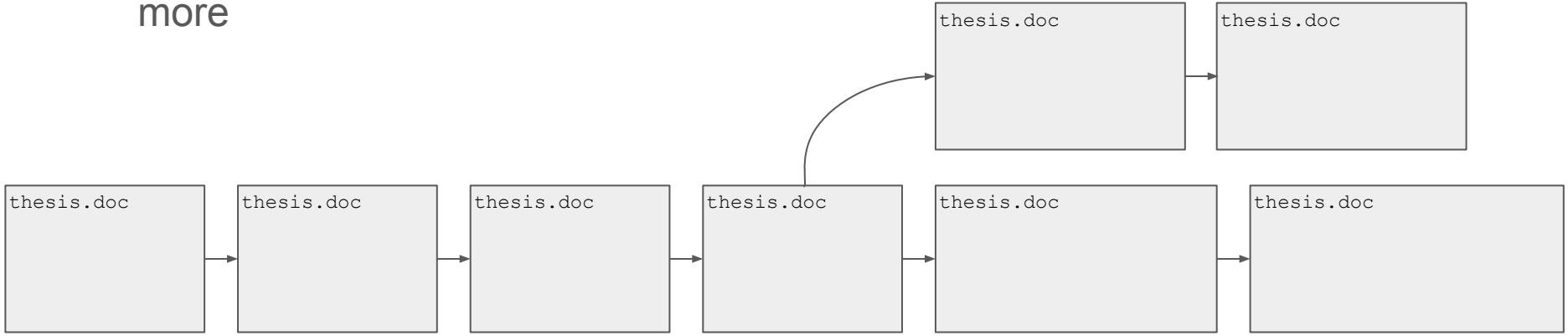
- Tree-like structure



Git -- "Version Control System" (VCS)

How about we organize things:

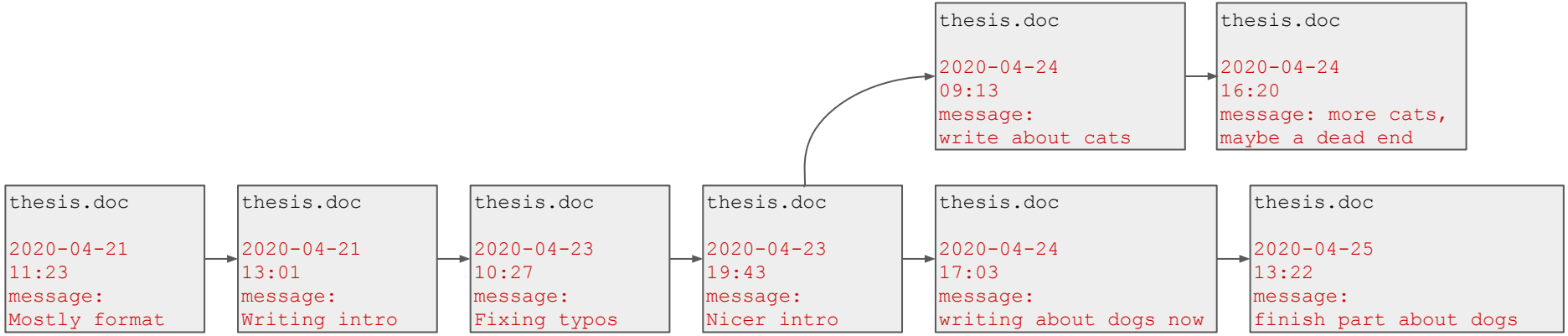
- We don't need different file names any more



Git -- "Version Control System" (VCS)

How about we organize things:

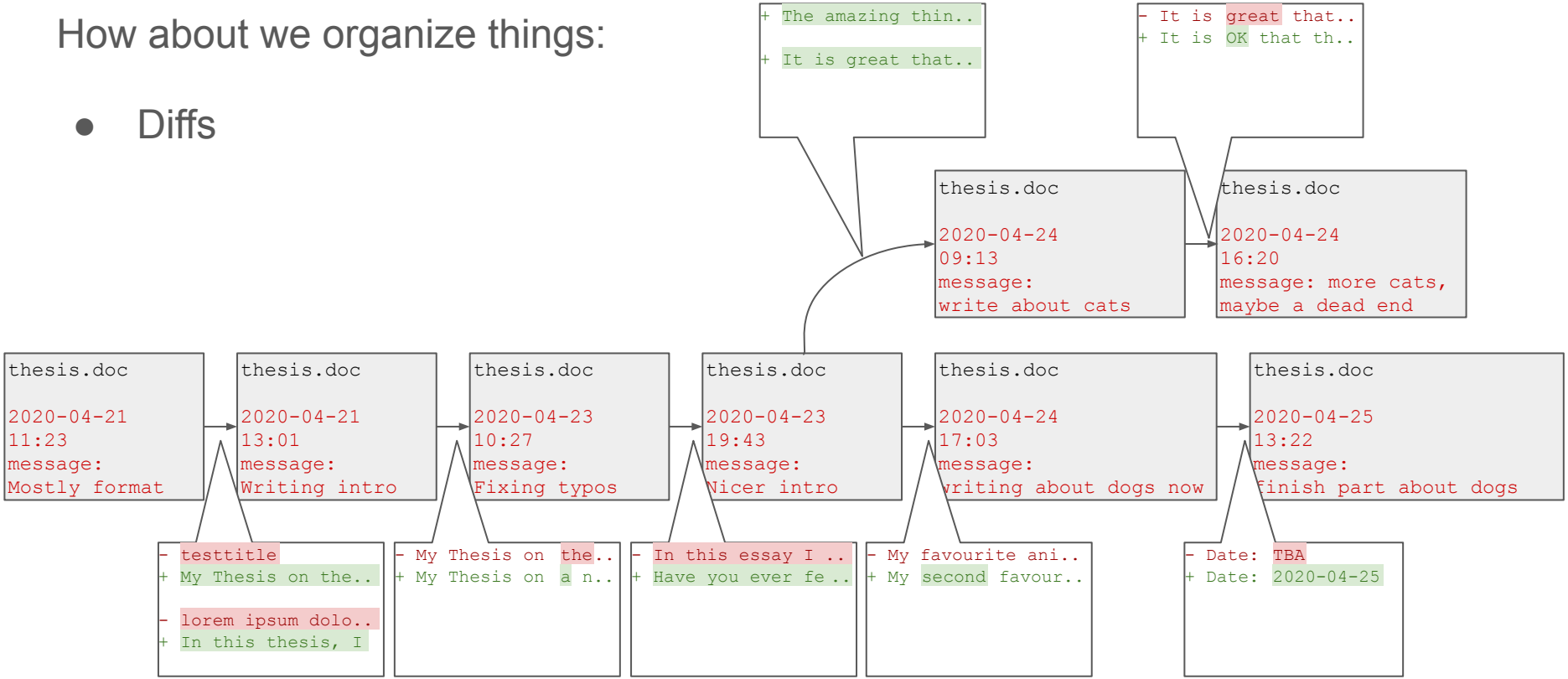
- Metadata



Git -- "Version Control System" (VCS)

How about we organize things:

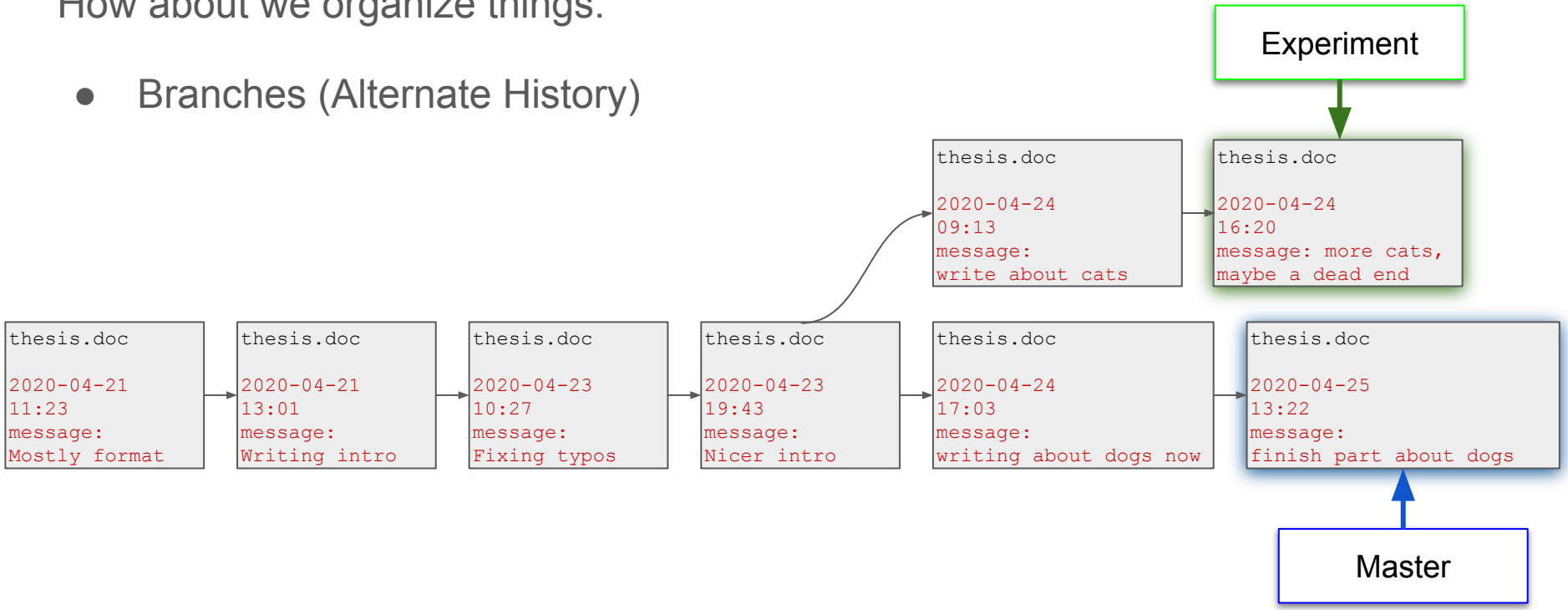
- Diffs



Git -- "Version Control System" (VCS)

How about we organize things:

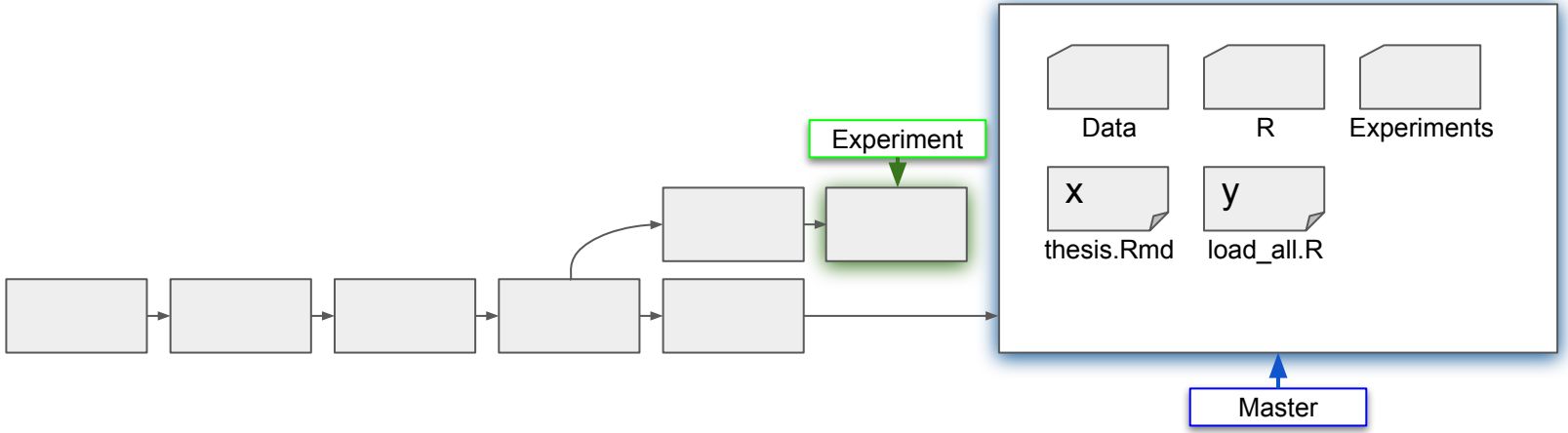
- Branches (Alternate History)



Git -- "Version Control System" (VCS)

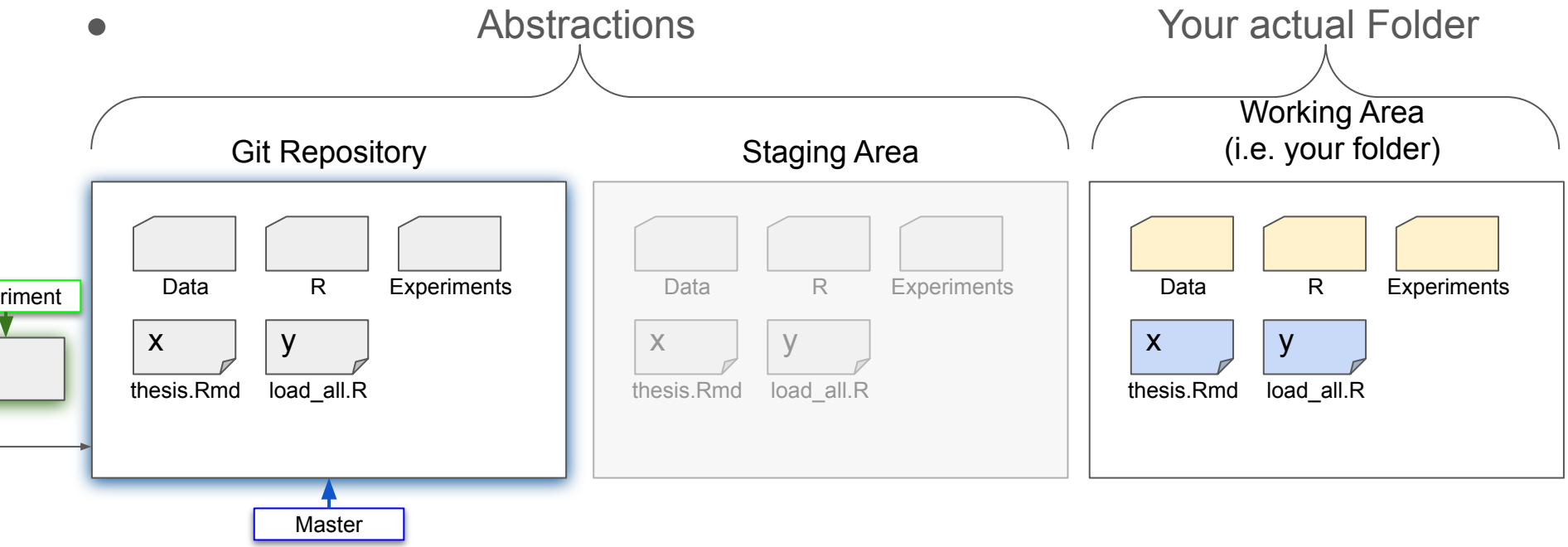
How about we organize things:

- More than one File



Git -- "Version Control System" (VCS)

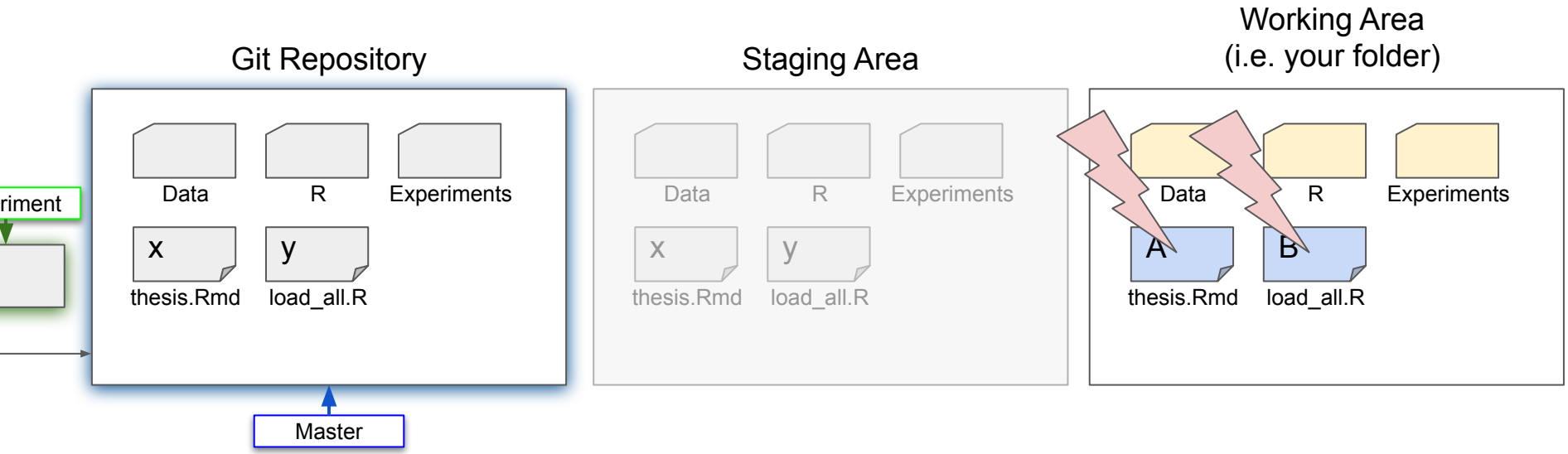
How about we organize things:

- 

Git -- "Version Control System" (VCS)

How about we organize things:

- Changing Files

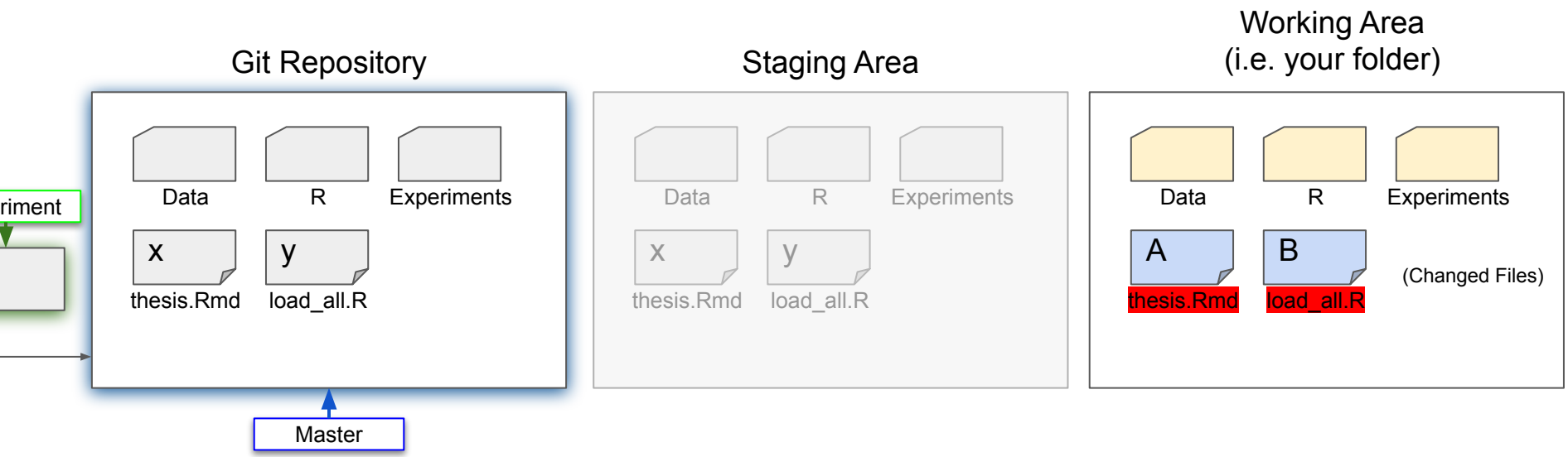


Git -- "Version Control System" (VCS)

How about we organize things:

- Changing Files: `> git status`

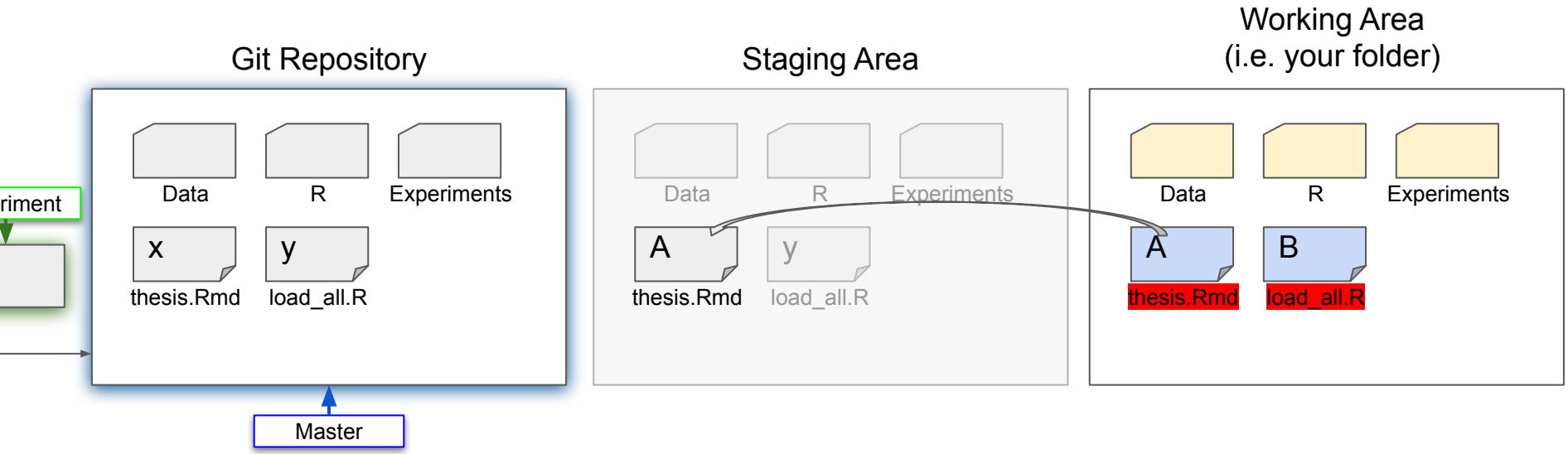
```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will
  (use "git checkout -- <file>..." to discard c

        modified:   load_all.R
        modified:   thesis.Rmd
```



Git -- "Version Control System" (VCS)

How about we organize things:

- "Staging" Files: `> git add <filename>`

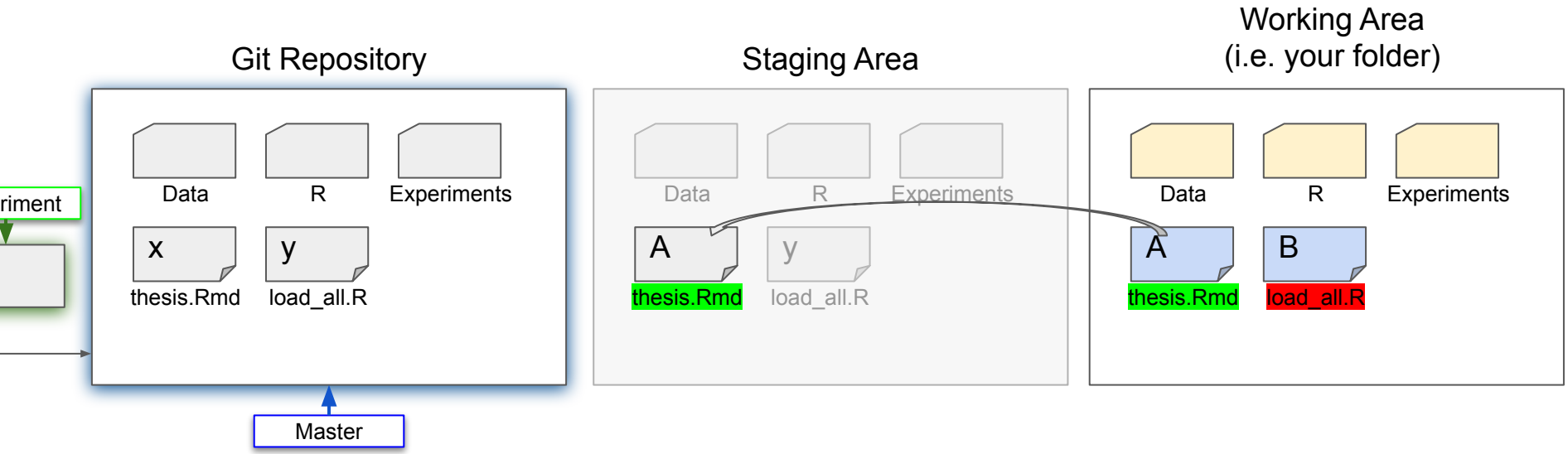


Git -- "Version Control System" (

How about we organize things:

- "Staging" Files: `> git status`

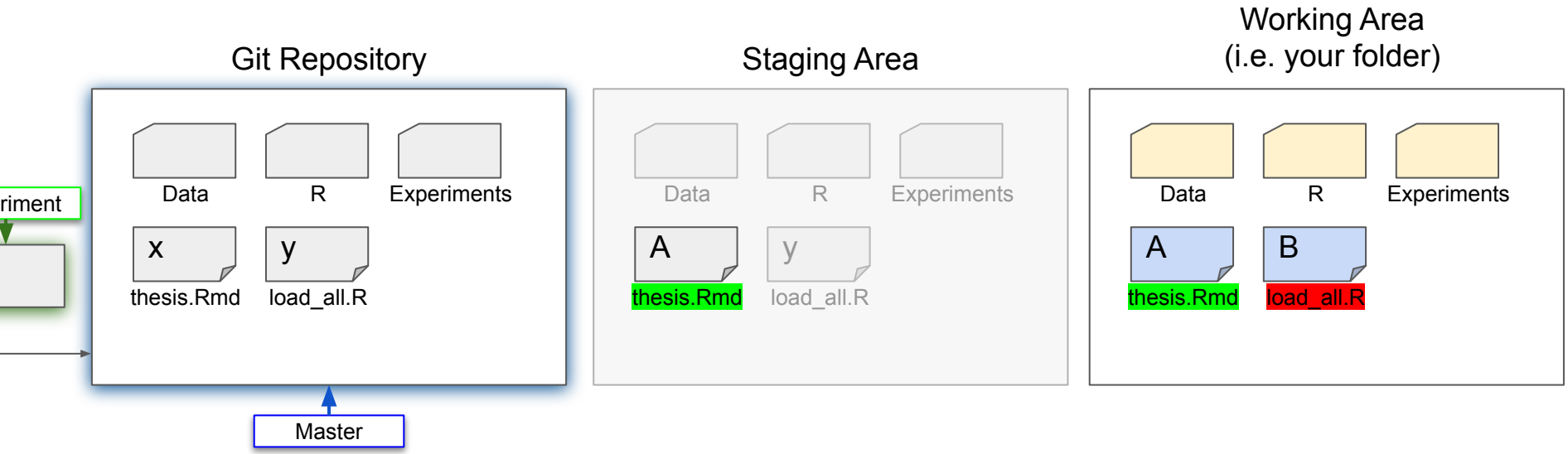
```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   thesis.Rmd

Changes not staged for commit:
  (use "git add <file>..." to update what will
  (use "git checkout -- <file>..." to discard c

        modified:   load_all.R
```



Git -- "Version Control System" (VCS)

How about we organize things:

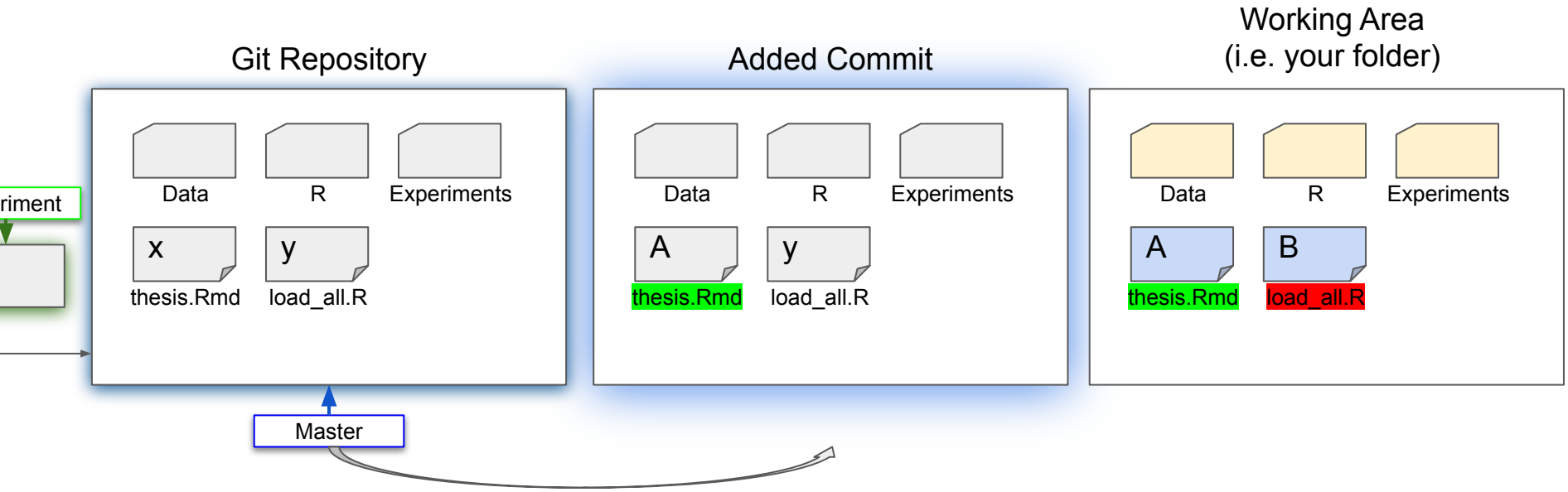
- "Commit" Files: `> git commit`



Git -- "Version Control System" (VCS)

How about we organize things:

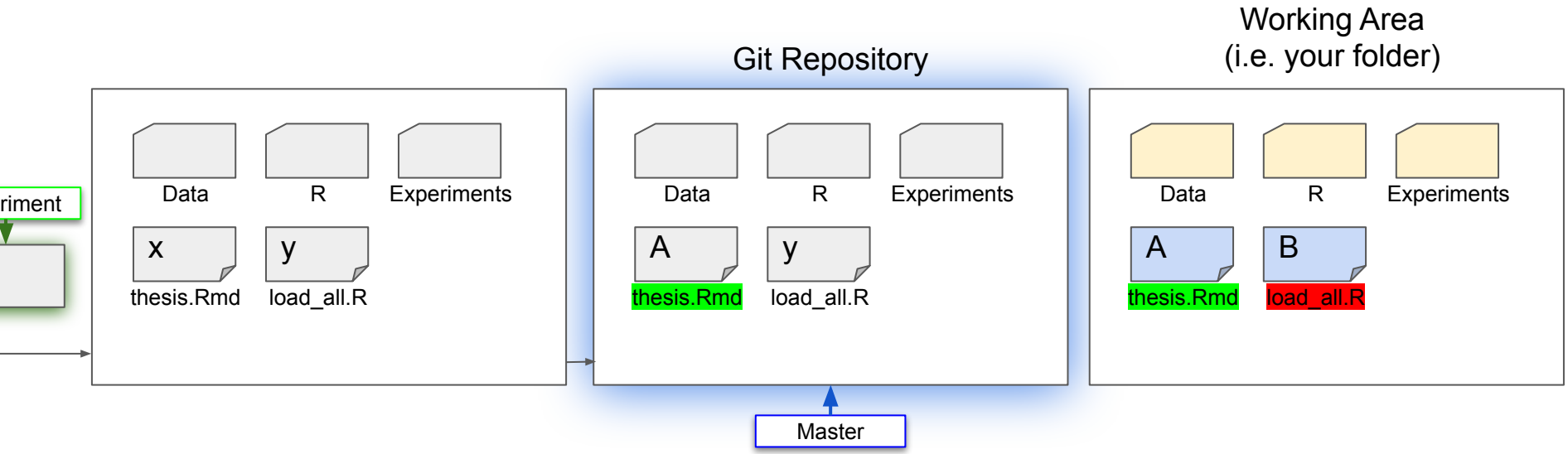
- "Commit" Files: `> git commit`



Git -- "Version Control System" (VCS)

How about we organize things:

- "Commit" Files: `> git commit`

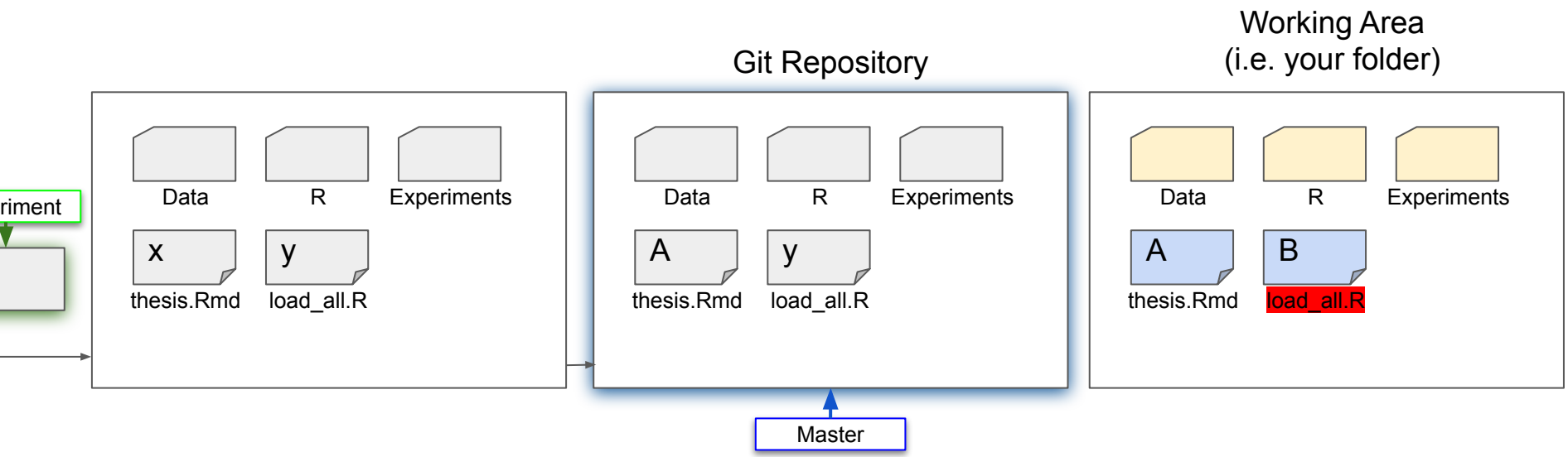


Git -- "Version Control System" (VCS)

How about we organize things:

- Status now: `> git status`

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will
  (use "git checkout -- <file>..." to discard c

        modified:   load_all.R
```

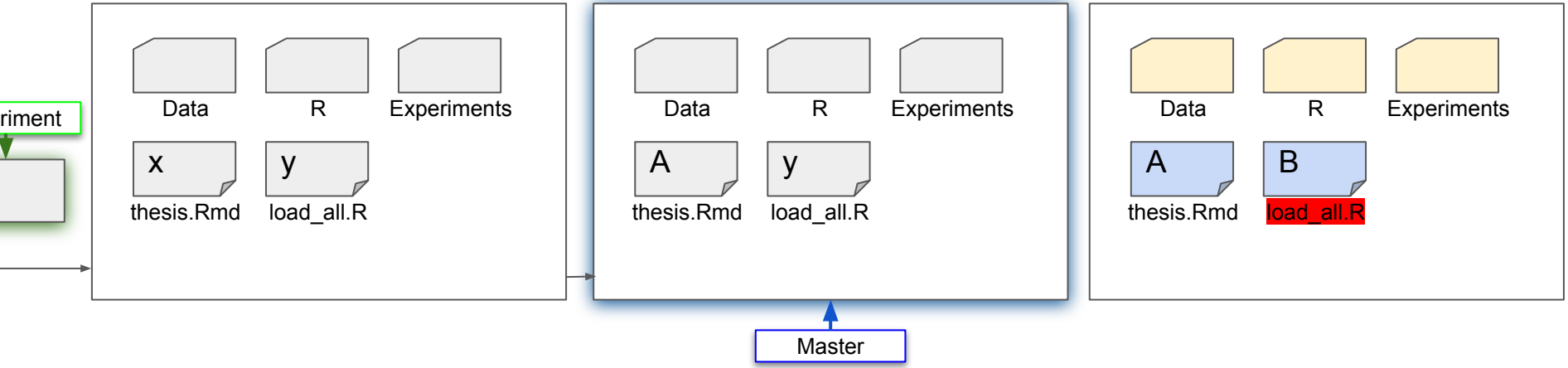


Git -- "Version Control System" (VCS)

How about we organize things:

- Changed lines: `> git diff`

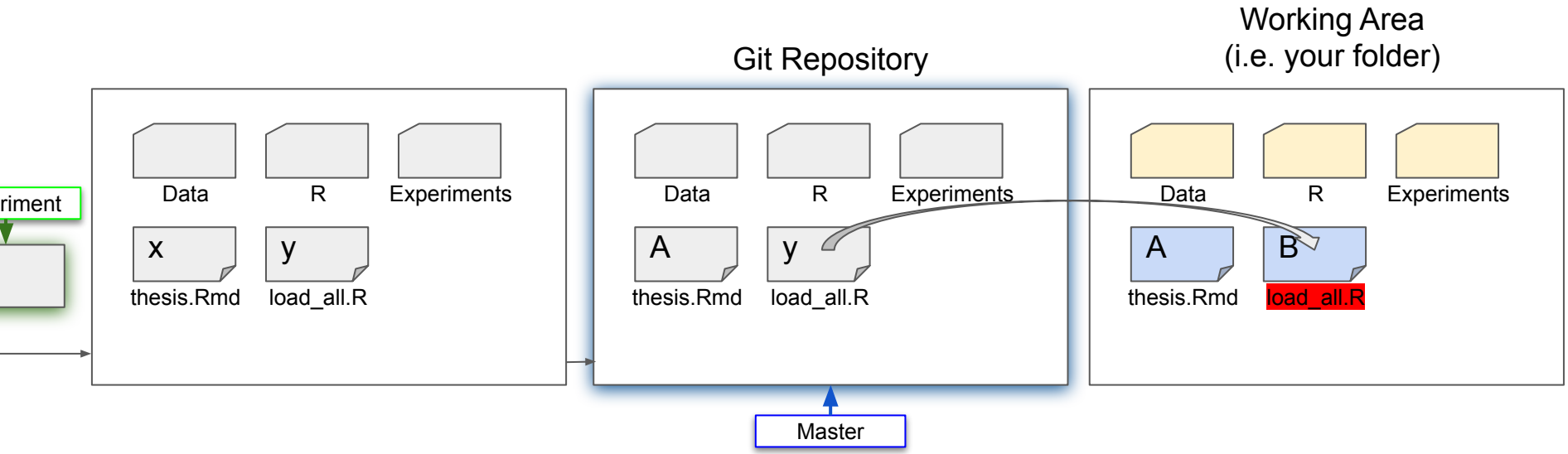
```
diff --git a/load_all.R b/load_all.R
index 975fbec..223b783 100644
--- a/load_all.R
+++ b/load_all.R
@@ -1 +1 @@
-y
+B
```



Git -- "Version Control System" (VCS)

How about we organize things:

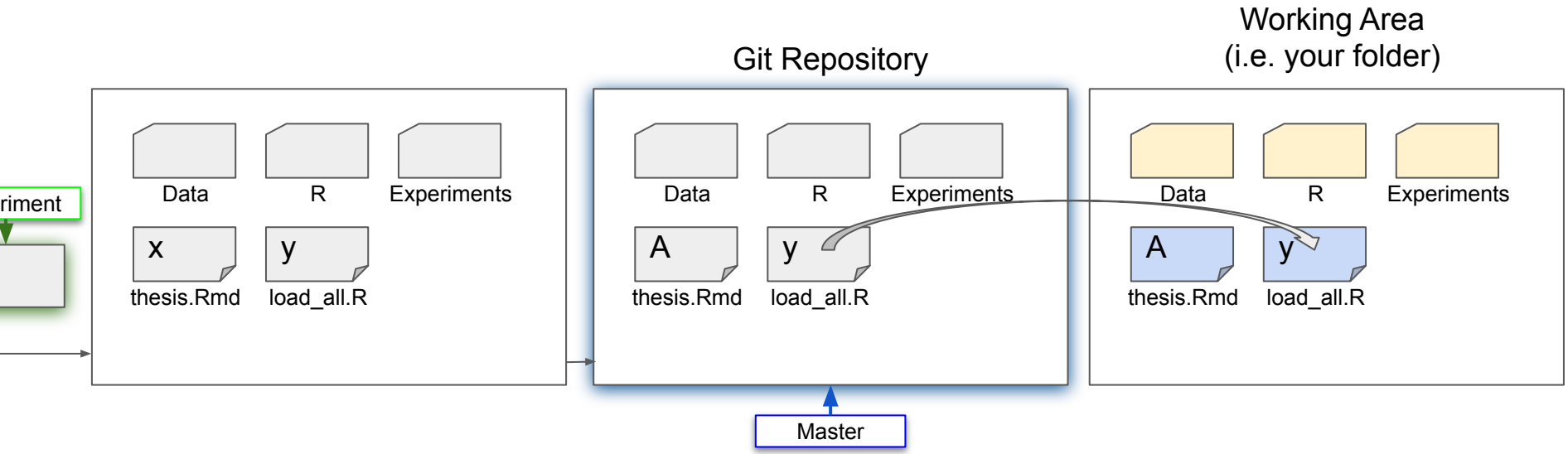
- "check out" Files: `> git checkout load_all.R`



Git -- "Version Control System" (VCS)

How about we organize things:

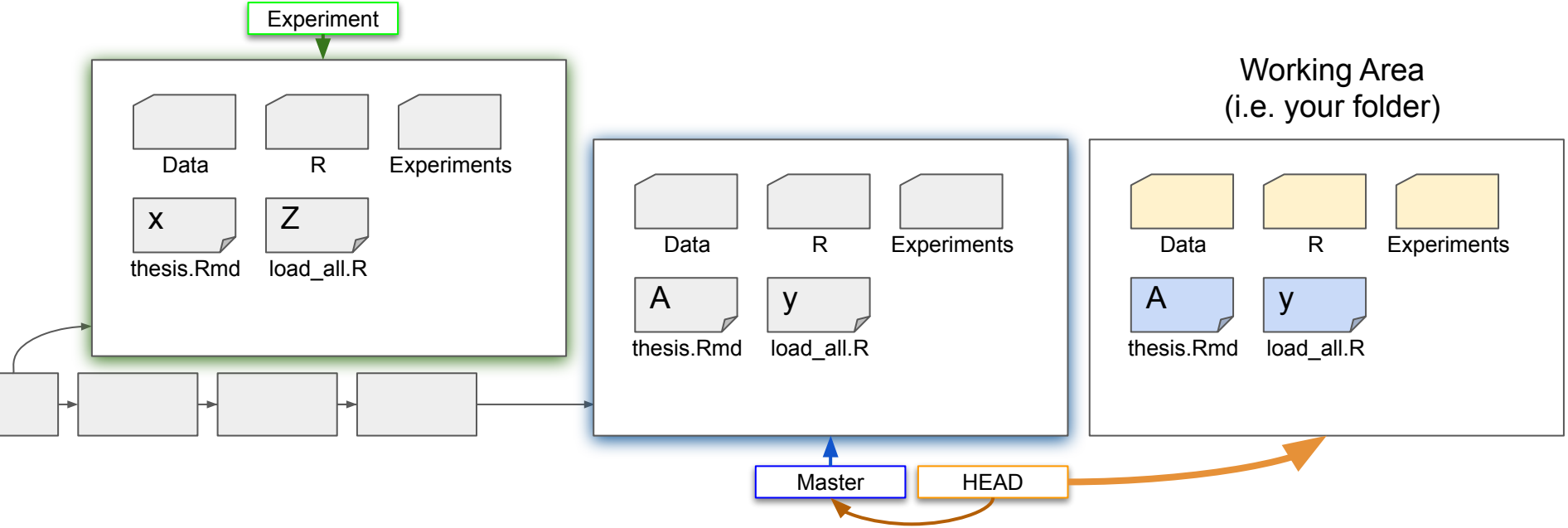
- "check out" Files: `> git checkout load_all.R`



Git -- "Version Control System" (VCS)

How about we organize things:

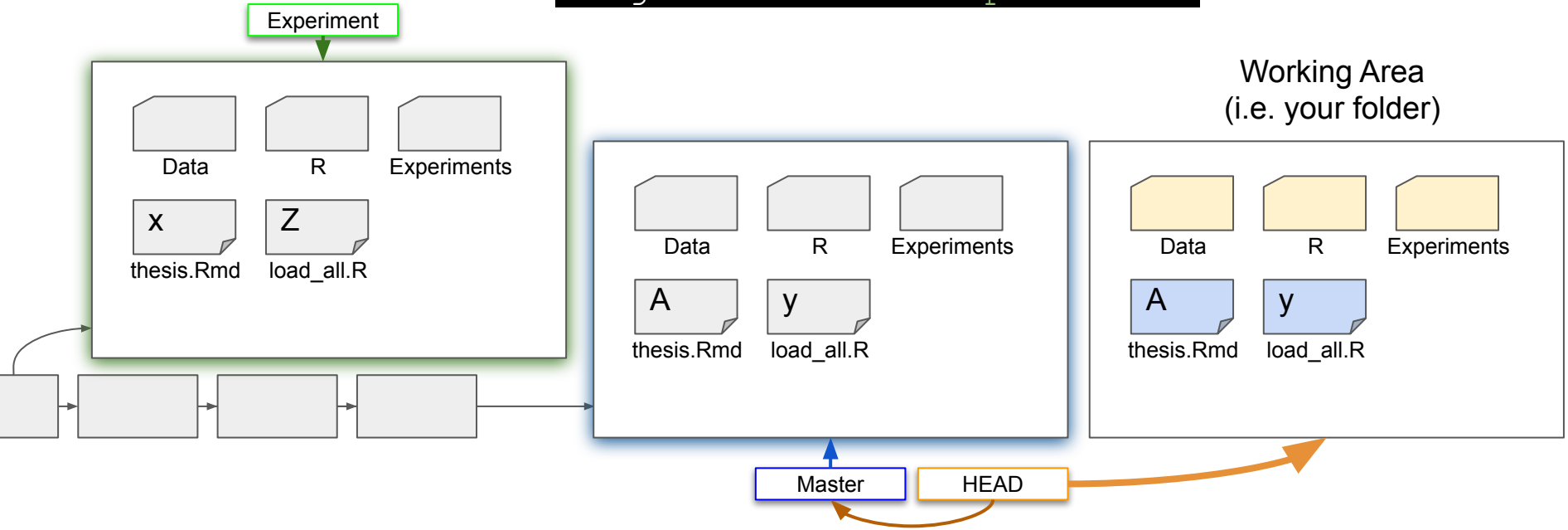
- different Branches



Git -- "Version Control System" (VCS)

How about we organize things:

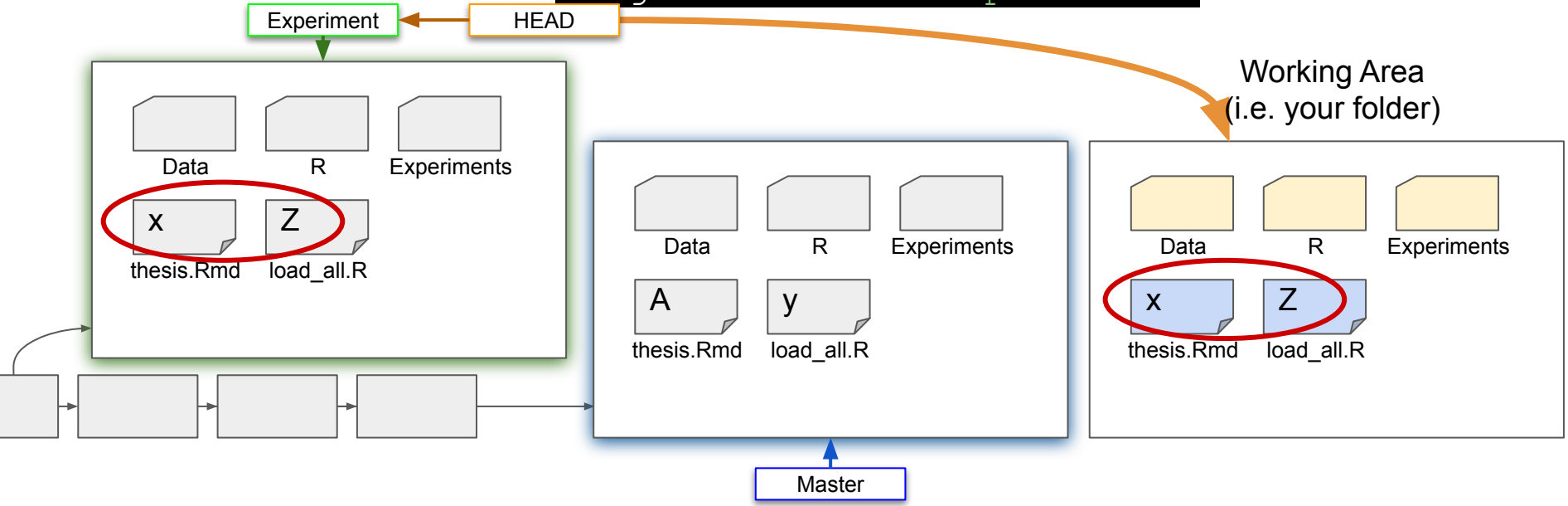
- "check out" *branches*: `> git checkout Experiment`



Git -- "Version Control System" (VCS)

How about we organize things:

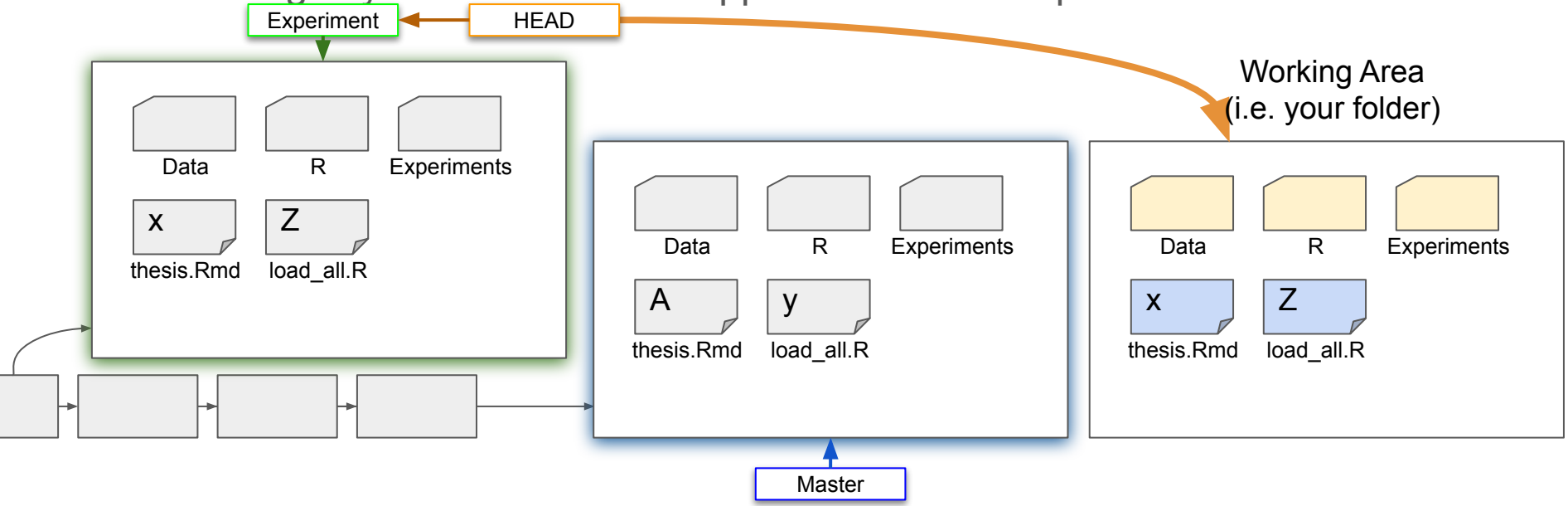
- "check out" *branches*: `> git checkout Experiment`



Git -- "Version Control System" (VCS)

How about we organize things:

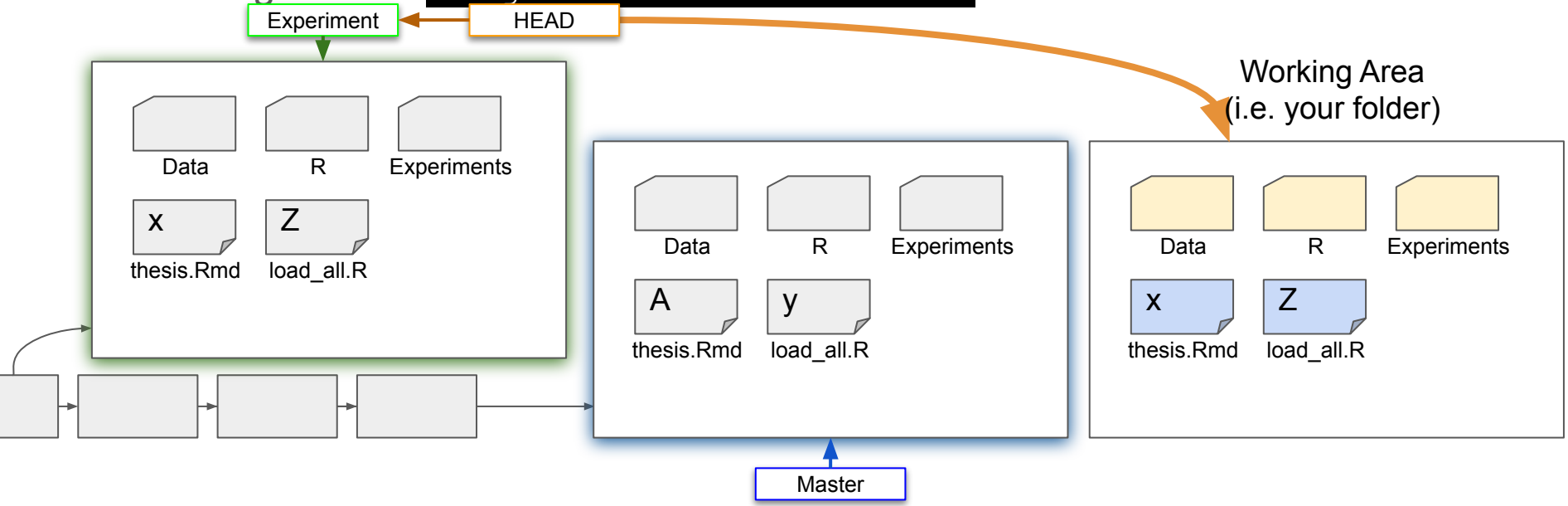
- all changes you make are now appended to the "Experiment" branch!



Git -- "Version Control System" (VCS)

How about we organize things:

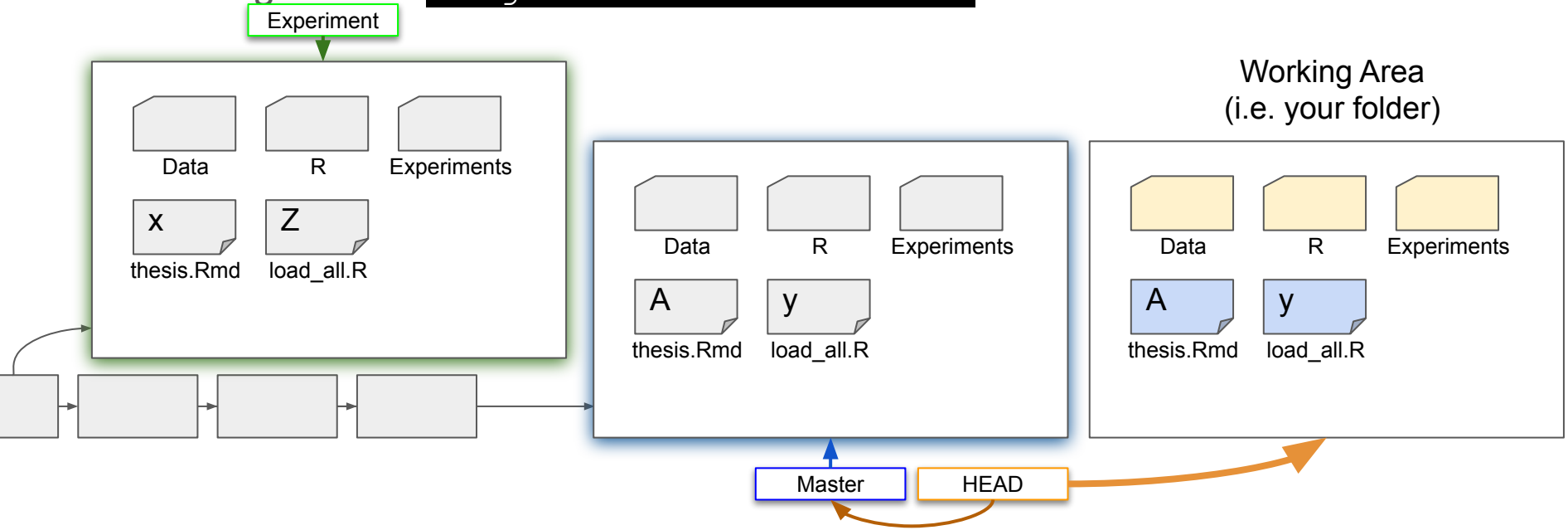
- Let's go back: `> git checkout master`



Git -- "Version Control System" (VCS)

How about we organize things:

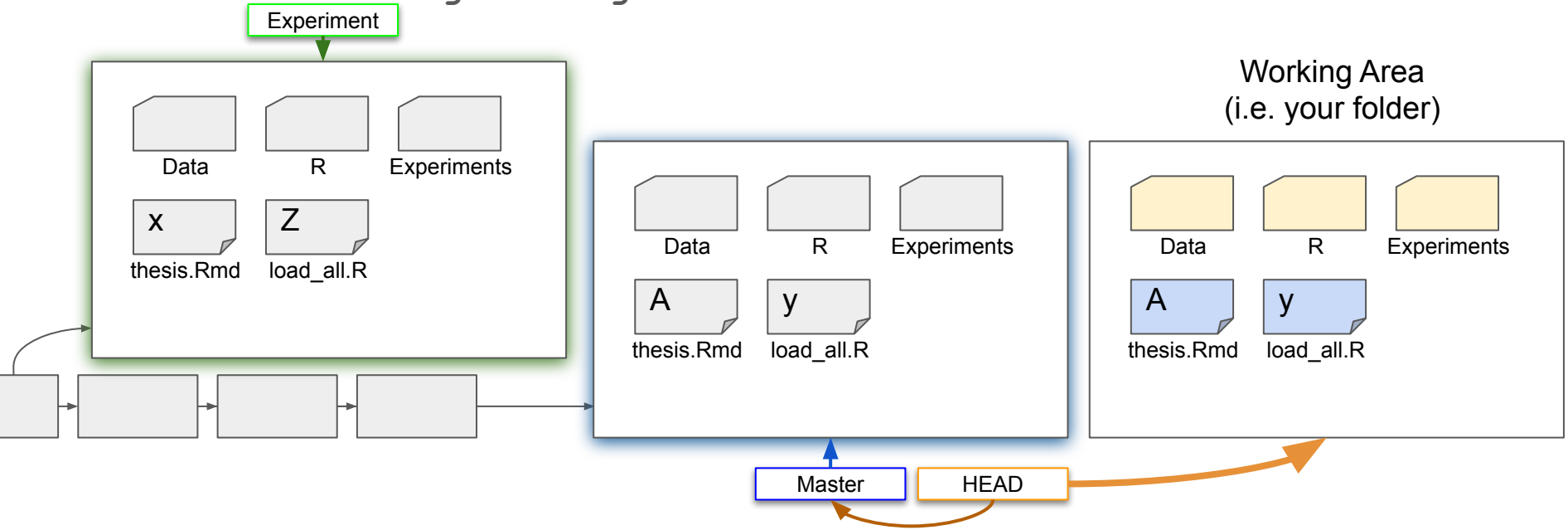
- Let's go back: `> git checkout master`



Git -- "Version Control System" (VCS)

How about we organize things:

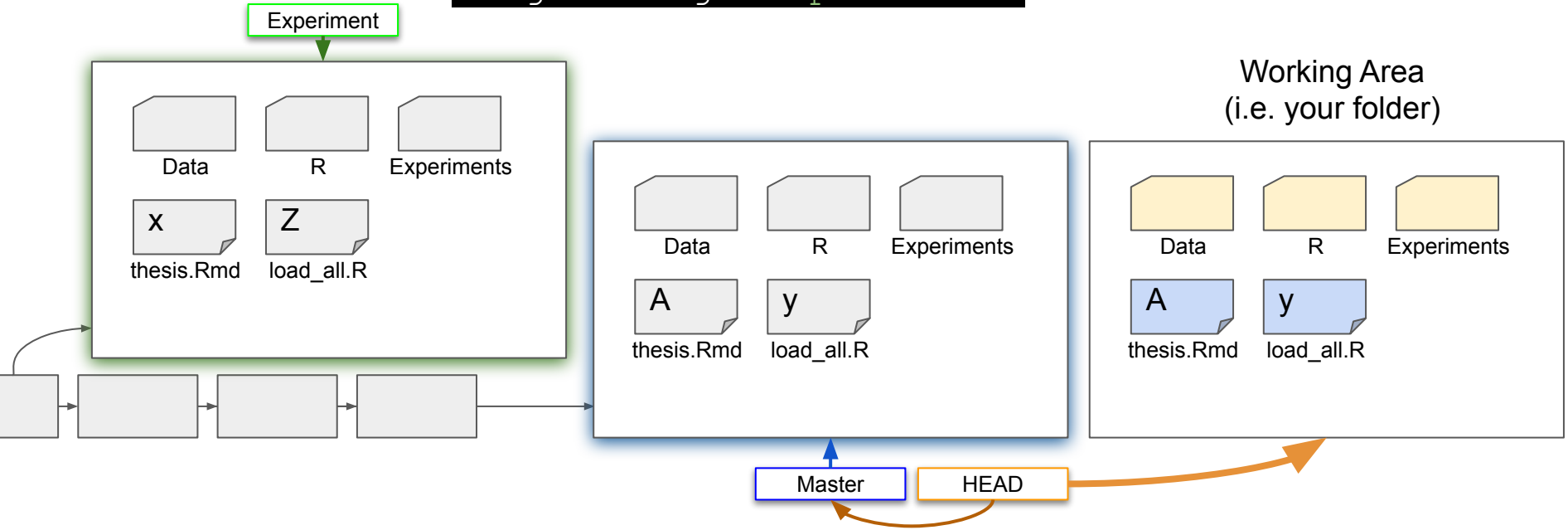
- Unite Histories: `git merge`



Git -- "Version Control System" (VCS)

How about we organize things:

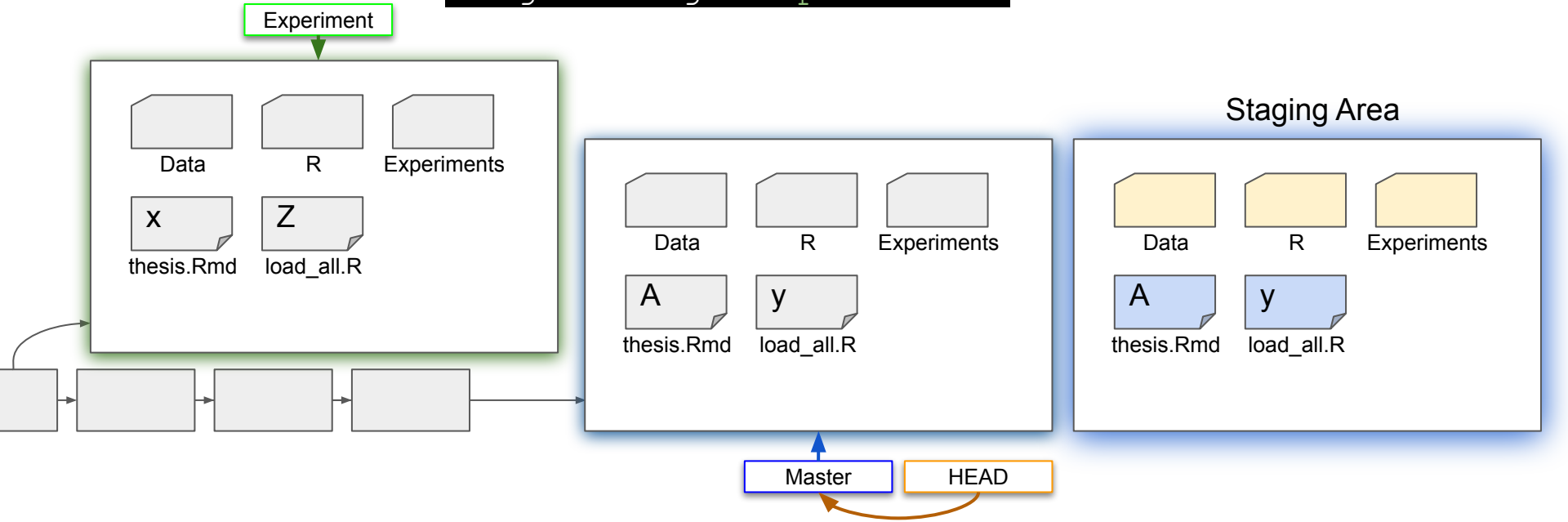
- Unite Histories: `> git merge Experiment`



Git -- "Version Control System" (VCS)

How about we organize things:

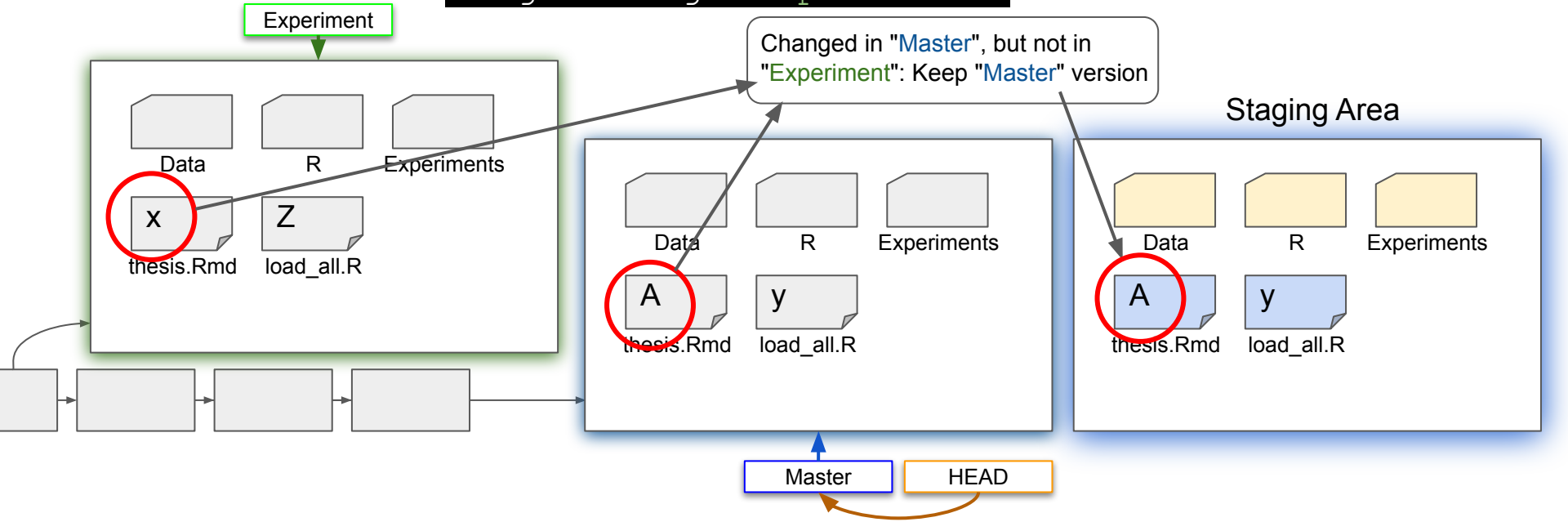
- Unite Histories: `> git merge Experiment`



Git -- "Version Control System" (VCS)

How about we organize things:

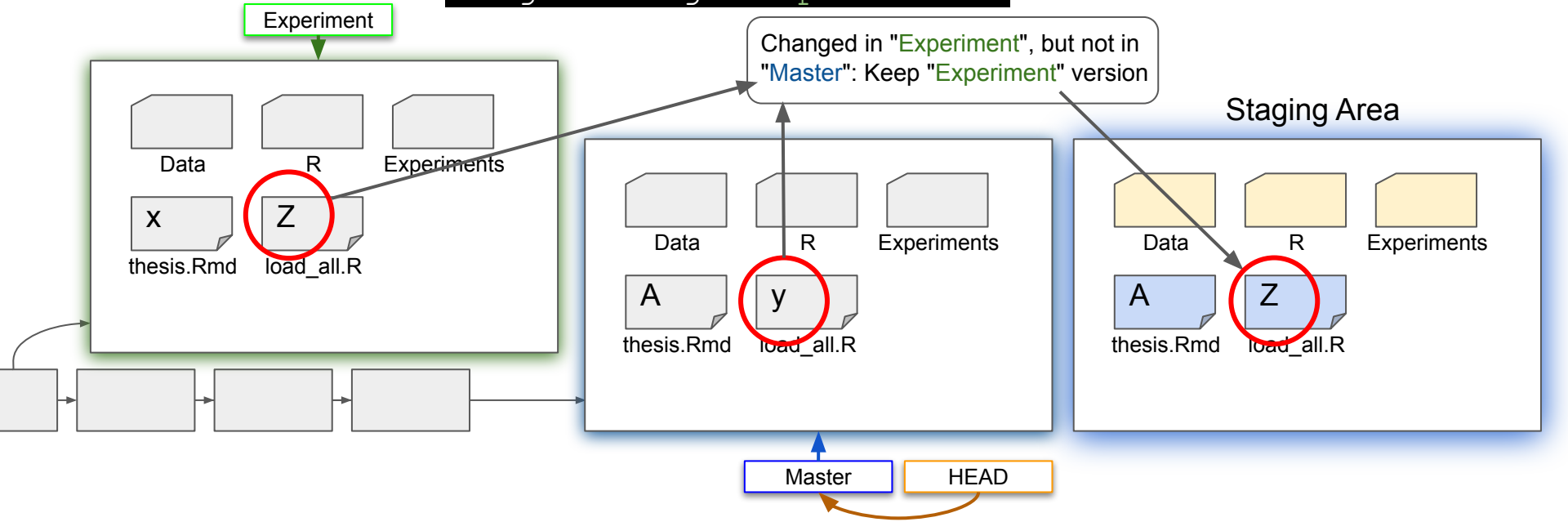
- Unite Histories: `> git merge Experiment`



Git -- "Version Control System" (VCS)

How about we organize things:

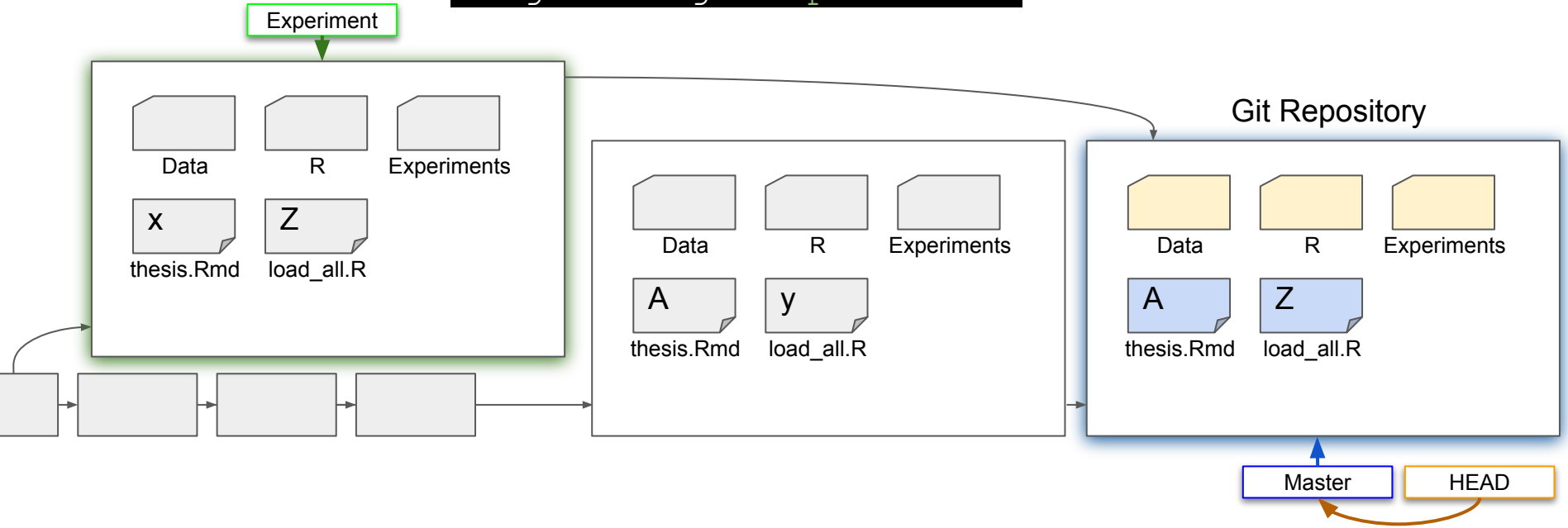
- Unite Histories: `> git merge Experiment`



Git -- "Version Control System" (VCS)

How about we organize things:

- Unite Histories: `> git merge Experiment`

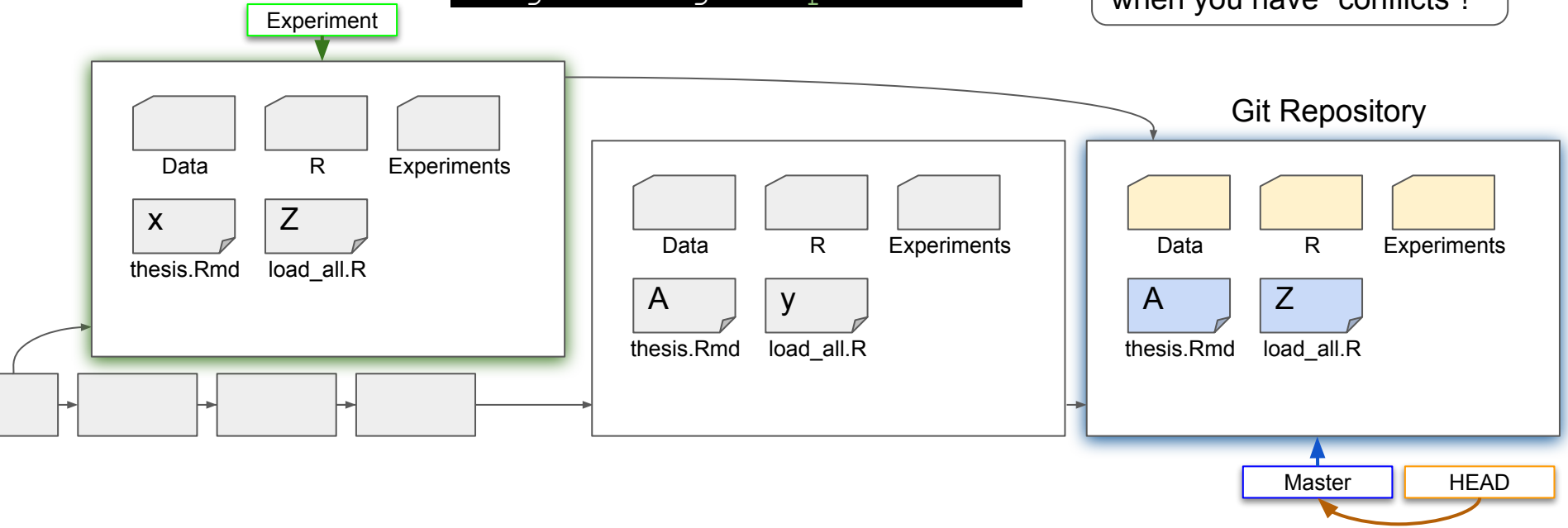


Git -- "Version Control System" (VCS)

How about we organize things:

- Unite Histories: `> git merge Experiment`

Not always this smooth when you have "conflicts"!



Git -- "Version Control System" (VCS)

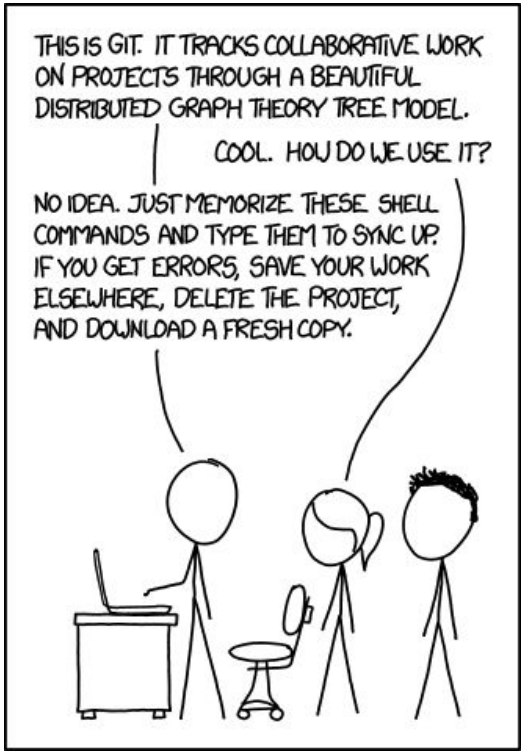
- You should now see that **git** is very useful
- Learning git is notoriously difficult, but you certainly get what you pay for
- Features of git we expect you to learn:
  - simple version control -- keep track of changed files
  - branching for alternate history of files
  - "merging" of branches to combine edits done in these branches
  - Not shown here: Using git for collaborative work
    - "`fetch`" / "`pull`" from *remote repositories*
    - "`push`" to remote repositories
    - You will learn this in your homework!

What We Expect You to Know

git commands to know:

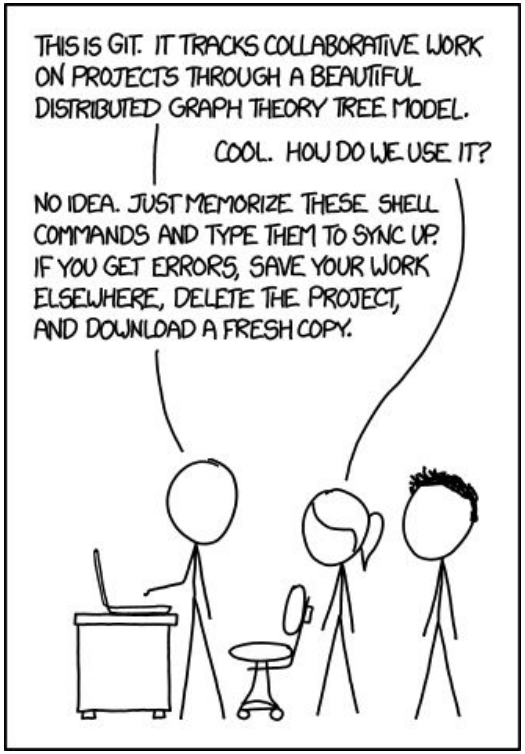
- `git init` -- create a new (empty) repository
- `git clone` -- create a repository linked to another (online) repo
- `git status` -- see if files have changed
- `git checkout` -- check out files or branches
  - `git checkout -b "new_branch_name"` -- create new branch
- `git branch` -- find out on what branch you are
- `git merge` -- take changes from another branch into the current one
  - know how to handle merge conflicts!
- `git diff` -- see what has changed (default: since last commit)
- `git add` -- add changes to the staging area to be committed
- `git commit` -- commit changes in the staging area
  - `git commit -a` -- commit *all* changes
- `git fetch` -- get changes from remote repo and DON'T merge
- `git pull` -- get changes from remote repo and DO merge
- `git push` -- push changes to remote repo
- The `.gitignore` file -- let git ignore certain files / file types

Git



https://xkcd.com/1597/

Git



https://xkcd.com/1597/

(<-- Please be better than this)