Introduction and overview

What is “software engineering”?

- **Software DESIGN:** creating the high-level architecture of your code, choosing the optimal data structures and algorithms based on the specific applications and limitations

- **Software DEVELOPMENT:** writing, testing, and debugging your code, consistent with the software design that has been laid out

- **Software MAINTENANCE:** organizing, modifying, updating, and further documenting your code once it has been released

For this week…

- ~~**Software DESIGN:** creating the high-level architecture of your code, choosing the optimal data structures and algorithms based on the specific applications and limitations~~

- **Software DEVELOPMENT:** writing, testing, and debugging your code, consistent with the software design that has been laid out

- **Software MAINTENANCE:** organizing, modifying, updating, and further documenting your code once it has been released

There is a big need for rigorous software reproducibility

nature

Explore content | Journal information | Publish with us | Subscribe

nature > technology features > article

TECHNOLOGY FEATURE | 24 August 2020

Challenge to scientists: does your ten-year-old code still run?

Missing documentation and obsolete environments force participants in the Ten Years Reproducibility Challenge to get creative.

Jeffrey M. Perkel

PLOS | COMPUTATIONAL BIOLOGY

EDITORIAL

Ten Simple Rules for Taking Advantage of Git and GitHub

Yasset Perez-Riverol[1]*, Laurent Gatto[2], Rui Wang[1], Timo Sachsenberg[3], Julian Uszkoreit[4], Felipe da Veiga Leprevost[5], Christian Fufezan[6], Tobias Ternent[1], Stephen J. Eglen[7], Daniel S. Katz[8], Tom J. Pollard[9], Alexander Konovalov[10], Robert M. Flight[11], Kai Blin[12], Juan Antonio Vizcaíno[1]*

PROTOCOLS article

Front. Neuroinform., 04 January 2018 | https://doi.org/10.3389/fninf.2017.00069

Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions

Fabien C. Y. Benureau[1,2,3*] and Nicolas P. Rougier[1,2,3]

[1]INRIA Bordeaux Sud-Ouest, Talence, France

[2]Institut des Maladies Neurodégénératives, Université de Bordeaux, Centre National de la Recherche Scientifique UMR 5293, Bordeaux, France

[3]LaBRI, Université de Bordeaux, Bordeaux INP, Centre National de la Recherche Scientifique UMR 5800, Talence, France

PLOS BIOLOGY

OPEN ACCESS

PERSPECTIVE

Challenges and recommendations to improve the installability and archival stability of omics computational tools

Serghei Mangul, Thiago Mosqueiro, Richard J. Abdill, Dat Duong, Keith Mitchell, Varuni Sarwal, Brian Hill, Jaqueline Brito, Russell Jared Littman, Benjamin Statz, Angela Ka-Mei Lam, Gargi Dayama, Laura Grieneisen, [...], Ran Blekhman [view all]

https://doi.org/10.1038/d41586-020-02462-7
https://doi.org/10.1371/journal.pcbi.1004947
https://doi.org/10.3389/fninf.2017.00069
https://doi.org/10.1371/journal.pbio.3000333

REPRODUCIBILITY CHECKLIST

Code Workflows based on point-and-click interfaces, such as Excel, are not reproducible. Enshrine your computations and data manipulation in code.

Document Use comments, computational notebooks and README files to explain how your code works, and to define the expected parameters and the computational environment required.

Record Make a note of key parameters, such as the 'seed' values used to start a random-number generator. Such records allow you to reproduce runs, track down bugs and follow up on unexpected results.

Test Create a suite of test functions. Use positive and negative control data sets to ensure you get the expected results, and run those tests throughout development to squash bugs as they arise.

Guide Create a master script (e.g. a 'run.sh' file) that downloads required data sets and variables, executes your workflow and provides an obvious entry point to the code.

Archive GitHub is a one option. Archiving services such as Zenodo and Software Heritage promise long-term stability.

Track Use version-control tools such as Git to record your project's history.

Package Create ready-to-use computational environments using containerization tools (for example, Docker, Singularity), web services (Code Ocean, Gigantum, Binder) or virtual-environment managers (Conda).

Automate Use continuous-integration services (ex. Travis CI) to automatically test your code over time, and in various computational environments.

Simplify Avoid niche or hard-to-install third-party code libraries that can complicate reuse.

Verify Check your code's portability by running it in a range of computing environments.

nature

Explore content | Journal information | Publish with us | Subscribe

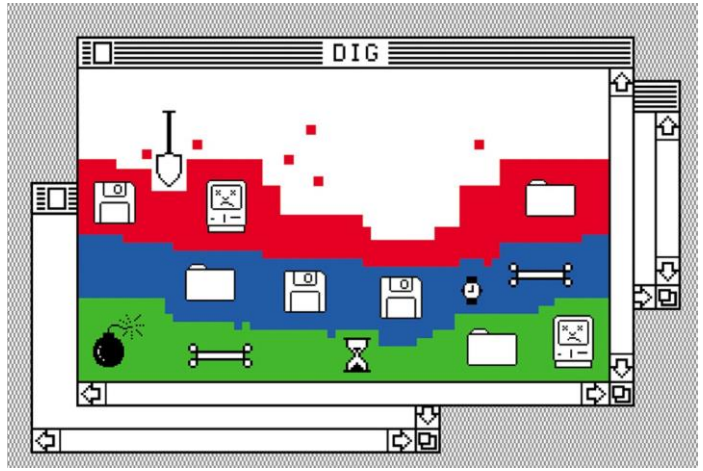
nature > technology features > article

TECHNOLOGY FEATURE | 24 August 2020

Challenge to scientists: does your ten-year-old code still run?

Missing documentation and obsolete environments force participants in the Ten Years Reproducibility Challenge to get creative.

Jeffrey M. Perkel



"Of the 43 articles they proposed reproducing, 28 resulted in reproducibility reports…"

The three pillars of the workshop…

I. Readable, robust, and reproducible code

PEP 8

II. Version Control

git

GitHub

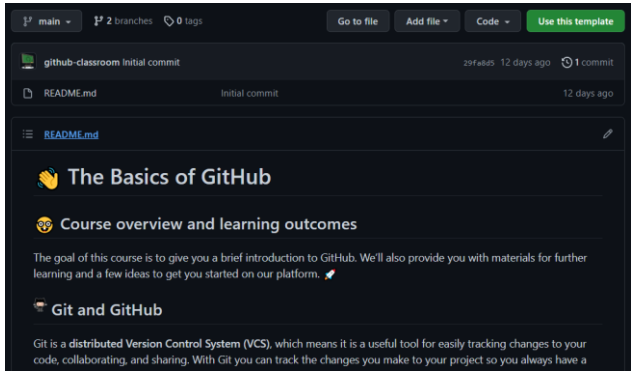
III. Test-driven development and optimization

Python UnitTest

Travis CI

(Jacob’s) coding best practices for Python

1. Follow the PEP8 style-guide
2. Write modularly
3. Comment extensively
4. Handle exceptions
5. Use argparse/configparse for input handling
6. Include thorough and consistent documentation
7. Log your errors
8. Write informative README’s



The Zen of Python

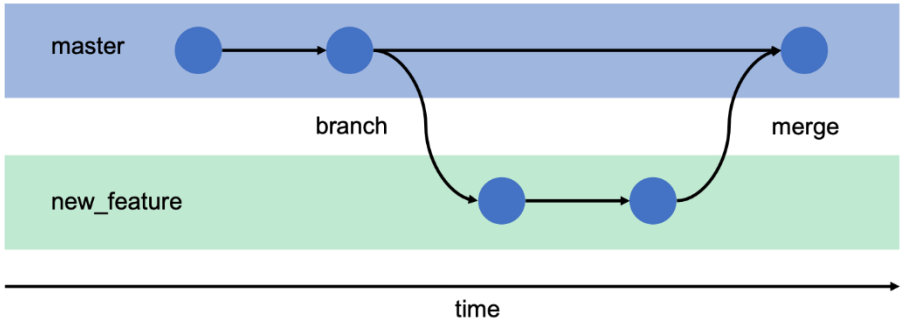
```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Version Control

A process by which you track and document the incremental changes of your software over the course of development

```
$ TZ=PST8PDT git log-compact --decorate --graph -n 17 v2.6.1
  === 2015-09-28 ===
* 22f698cb 19:19 jch (tag: v2.6.1) Git 2.6.1
*   3adc4ec7 19:16 jch Sync with v2.5.4
|\
| * 24358560 15:34 jch (tag: v2.5.4) Git 2.5.4
| *   11a458be 15:33 jch Sync with 2.4.10
| |\
| | * a2558fb8 15:30 jch (tag: v2.4.10) Git 2.4.10
| | *   6343e2f6 15:28 jch Sync with 2.3.10
| | |\
| | | * 18b58f70 15:26 jch (tag: v2.3.10, maint-2.3) Git 2.3.10
| | | *   92cdfd21 14:59 jch Merge branch 'jk/xdiff-memory-limits' into maint-2.3
| | | |\
| | | | * 83c4d380 14:58 jk  merge-file: enforce MAX_XDIFF_SIZE on incoming files
| | | | * dcd1742e 14:57 jk  xdiff: reject files larger than ~1GB
| | | | * 3efb9880 14:57 jk  react to errors in xdi_diff
| | | * |   f2df3104 14:46 jch Merge branch 'jk/transfer-limit-redirection' into maint-2.3
| | | |\ \
| | | | | | === 2015-09-25 ===
| | | | * | b2581164 15:32 bb  http: limit redirection depth
| | | | * | f4113cac 15:30 bb  http: limit redirection to protocol-whitelist
| | | | * | 5088d3b3 15:28 jk  transport: refactor protocol whitelist code
| | | | | |   === 2015-09-28 ===
| | | * | |   df37727a 14:33 jch Merge branch 'jk/transfer-limit-protocol' into maint-2.3
| | | |\ \ \
| | | | |/ /
| | | | | /
| | | | |/
| | | |/|
| | | | | === 2015-09-23 ===
| | | | * 33cfccbb 11:35 jk  submodule: allow only certain protocols for submodule fetches
```

git



GitHub

```
assets/js/model.js
1144 +    # Alert user whether we have a match or not
1145 +    function doPrediction()
1146 +        if (model.match = true)
```

Mona Lisa 23 hours ago

This will always be true. Looks like you're creating an assignment rather than an equality check...

Suggested change

```
1120 -    if (model.match = true)
1121 +    if (model.match == true)
```

Commit suggestion | Add suggestion to batch

PS: We use spaces, not tabs around here 😀

Reply...

Resolve conversation

Releases / Tags

Latest release

v0.17.2

def84ab

0.17.2

spraints released this a month ago

- Use the git author as the TFS committer during `git tfs rcheckin` (#336) and `git tfs rcheckin --quick` (#357)
- Improve temporary workspace handling (#328, #372)
- Use libgit2sharp more and git-core less (#361)
- Bug fix for bare repositories (#352)
- Bug fix for crash during `git tfs clone` (#349)
- Bug fix for VS2008 (#362)
- Update libgit2sharp
- Improved release process (#333, #340)

Full diff

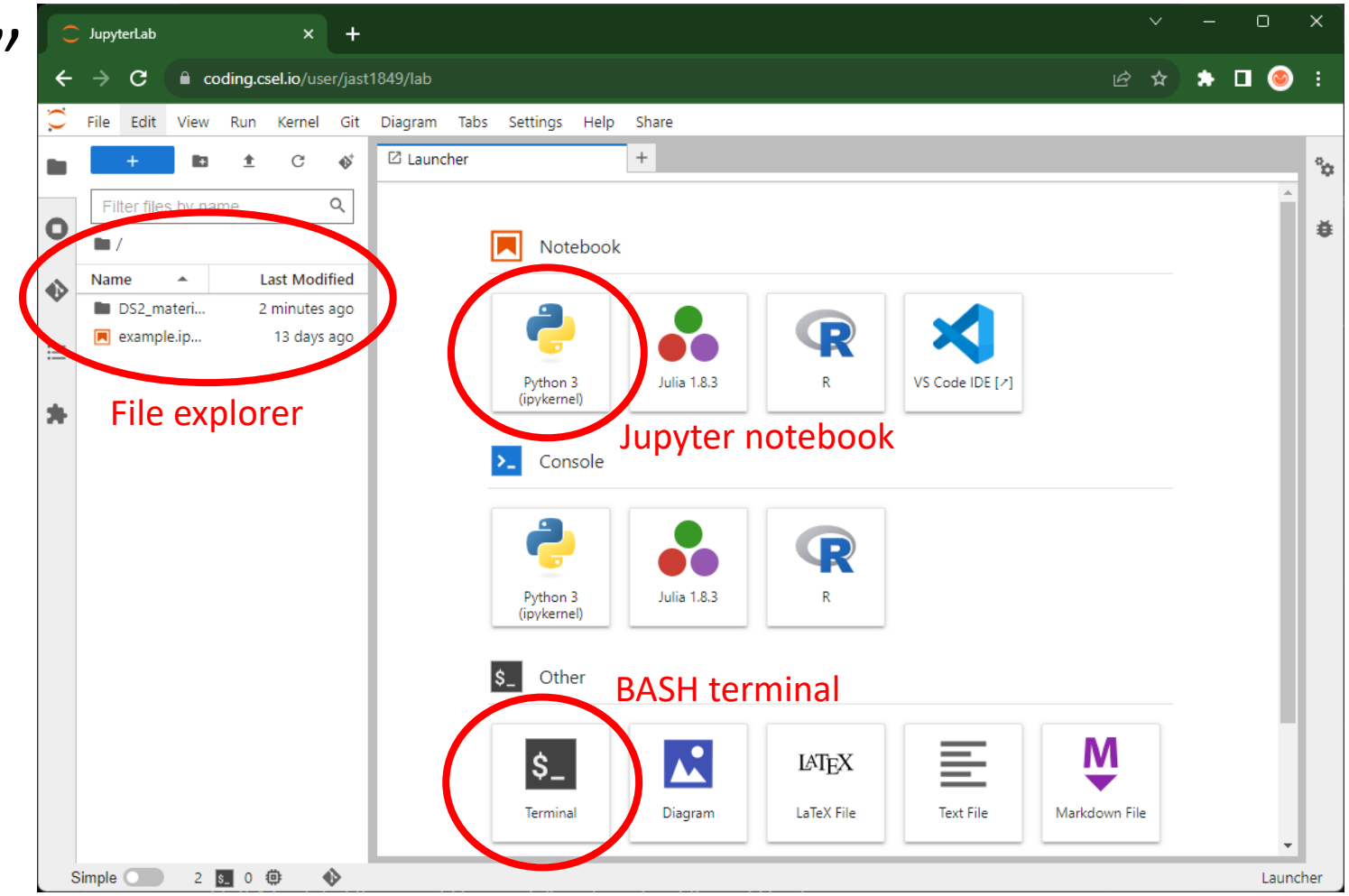
gittfs-0.17.2.zip | Source code (zip) | Source code (tar.gz)

Our “dev environment”

We will use a remote environment (CSEL) provided by the CS department.

Think of this like any other linux/unix computer system (like AWS, fiji, or your laptop)

Can be accessed at:
https://coding.csel.io/



University of Colorado Boulder

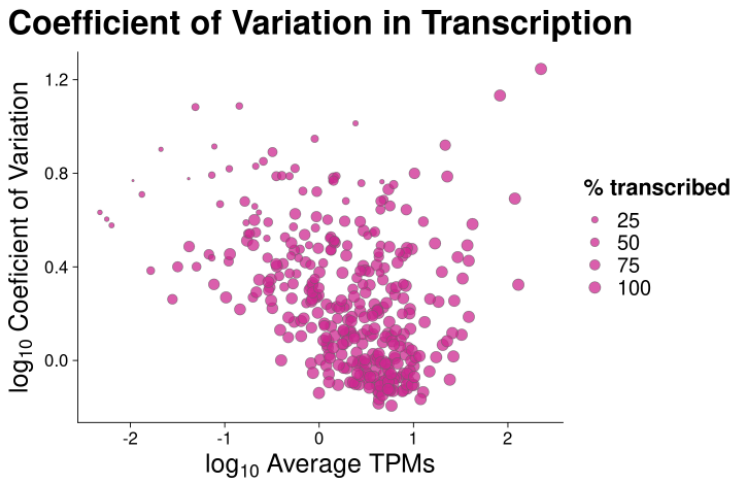
Coding @ Computer Science Education Lab

Department of Computer Science
University of Colorado, Boulder

Jupyter notebooks

Enable you to seamlessly combine live code with visualizations, equations, and narrative.

In [10]:
```
options(repr.plot.width=12, repr.plot.height=8)
txpt2 <- ggplot(hg38_tpms_hermit,
       aes(x=log(mean, base=10),
           y=log(coefvar, base=10),
          size=percent_transcribed)) +
ggtitle("Coefficient of Variation in Transcription") +
xlab(expression(paste(log[10],' Average TPMs'))) +
ylab(expression(paste(log[10], ' Coeficient of Variation', sep=""))) +
labs(size="% transcribed") +
geom_point(color = 'gray40',
           fill = 'maroon3',
           shape = 21,
           alpha = 0.75) +
theme_cowplot(24) +
theme(plot.title = element_text(hjust = 0.5),
      title = element_text(size = 34,  face = "bold"),
      axis.title = element_text(size = 30, face = "bold"),
      axis.text.x = element_text(size = 20),
      axis.text.y = element_text(size = 20),
     legend.title = element_text(size = 26),
     legend.text = element_text(size = 24))

txpt2
```



Normalized counts for genes in dbNascent

These are human counts for genes in human genome based on nascent trancription of 880 high QC samples.

- GC content with 150 bp of μ less that %50
- Less than %50 of bidirectionals that are in TSS regions
- Sample QC 1,2 and 3

The normaliztion was performed using counts from both *genes* and *bidirectionals* as follows:

$$\mathrm{TPM_i} = \frac{r_i/l_i}{\sum_1^j r_j/l_j} * 10^6$$

where r_i are the mapped reads for transcript i (for all genes and bidirectional transcripts), l_i is the transcript length and $\sum_j r_j/l_i$ sums all j length normalized transcripts. The ratio is multiplied by a scaling factor of 10^6. The counts for genes was normalized over the 5' truncated transcripts (750 bp at the 5' end of gene removed).

Lastly, genes with no transcription in **ALL** samples were also filtered out from this table.

In [3]:
```
hg38_tpms <- data.table::fread('/Users/rusi2317/projects/meta_analysis_qc/hg38/processed_data/final_co
unts/genes_5ptrunc_bidirectionals_tpm.tsv')
hg38_tpms$num_zeros <- rowSums(hg38_tpms[,6:885] == 0)
hg38_tpms$percent_zero_samples <- (hg38_tpms$num_zeros/880)*100
hg38_tpms$percent_transcribed <- 100-hg38_tpms$percent_zero_samples
dim(hg38_tpms)
head(hg38_tpms)
```

282165 · 896

A data.table: 6 × 896

chromosome	start	end	GeneID	Length	SRZ1950491	SRZ1950493	SRZ1950495	SRZ1950497	SRZ19
<chr>	<int>	<int>	<chr>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
chr1	77562416	77681908	ZZZ3	119493	5.89606855	8.4220380	6.065659968	6.340954238	6.4128
chr17	4004445	4142280	ZZEF1	137836	2.99959752	3.7204037	3.492450878	3.340299643	3.7528
chr7	143382095	143391111	ZYX	9017	1.34000571	0.9263288	0.368001010	0.500336067	0.8506
chr1	52727203	52827336	ZYG11B	100134	2.28827793	2.6288455	1.661646325	1.882821794	2.2627
chr1	52843510	52894995	ZYG11A	51486	0.02560166	0.0000000	0.009207122	0.009223826	0.0114
chr3	126458901	126475169	ZXDC	16269	4.48314983	5.1963559	4.705705105	4.101245248	5.3798

You can interact with these components modularly.

Our learning objectives

1. Get comfortable using Git + GitHub
2. Get comfortable using Jupyter notebooks
3. Practice Python basics and best practices
4. Learn the basics of testing and optimization

What will be covering this week?

Tentative schedule (topics subject to change)

Dates	Hour	Topic
Friday 7/28	1	Software engineering motivation/overview, CSEL, JupyterLab
	2	Version control, Git, GitHub intro
	3	Git, GitHub
Monday 7/31	1	Primer (variables, data types, whitespace, operations)
	2	Primer (functions, import, iterables, loops)
	3	Primer (if/then, str-parse, read files/stdin, cmd line variables)
Tuesday 8/1	1	Python best practices (PEP8/pycodestyle, modular coding, commenting/documentation)
	2	Python best practices (arg/configparser, exception handling, error logging)
	3	Python best practices (exception handling)
Wednesday 8/2	1	Common packages: numpy, scipy, pandas
	2	Unit testing
	3	Test driven development
Thursday 8/3	1	Optimization (timeit, getsizeof)
	2	Optimization (cProfile)
	3	GitHub revisited

Version Control

Version control manages changes to files for a project

```
def div(a, b):
  return a/b
```

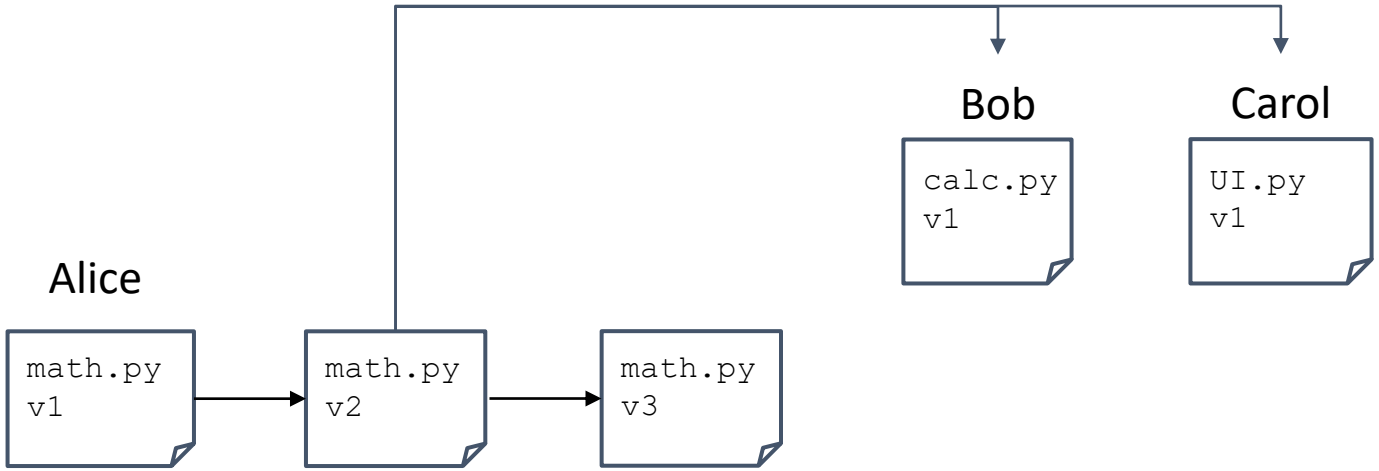
→

```
def div(a, b):
  if b > 0:
    return a/b
  else:
    return None
```

→

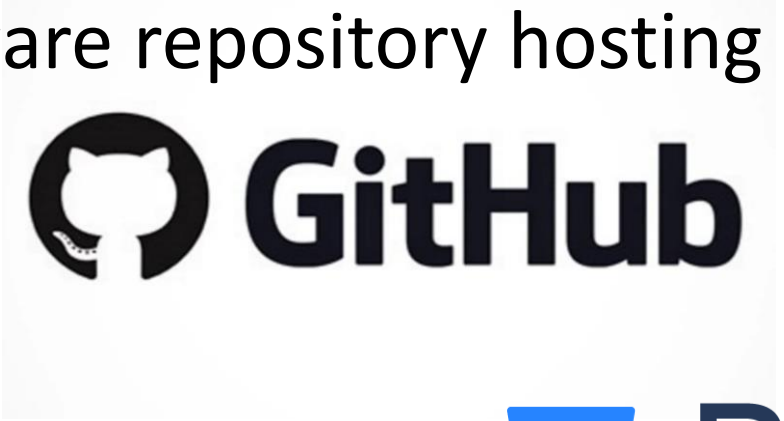
```
def div(a, b):
  if b > 0:
    return a/b
  else:
    return Inf
```

- reverting back to an old version
- allowing developers to test changes without losing the original
- synchronizing code between developers and users
- tagging specific versions



Version control software

Software repository hosting company



A couple definitions

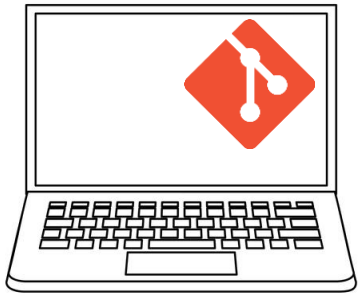
- **Repository ("repo"):** An archive containing all the saved states of all the files you are tracking.

- **"Commit":** The action of recording the state of your file(s) to the archive

Basic Git workflow

An itemized change to a file(s)

Hidden archival files that contain the files' full history



LOCAL repository

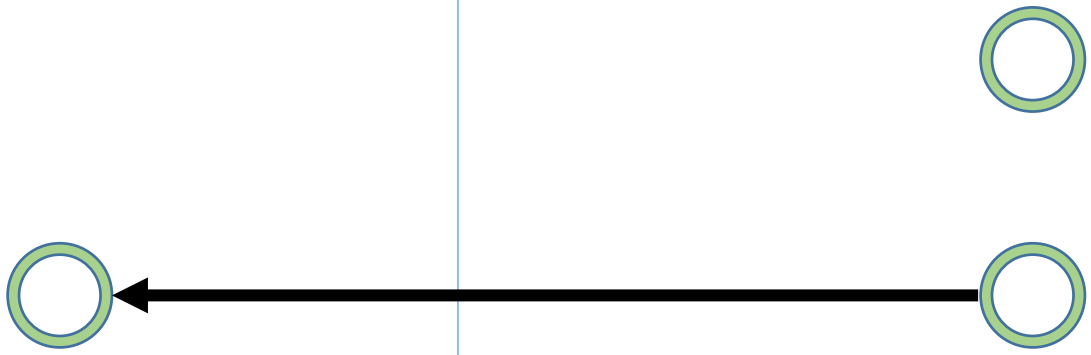
A collection of files you directly edit

LOCAL workspace

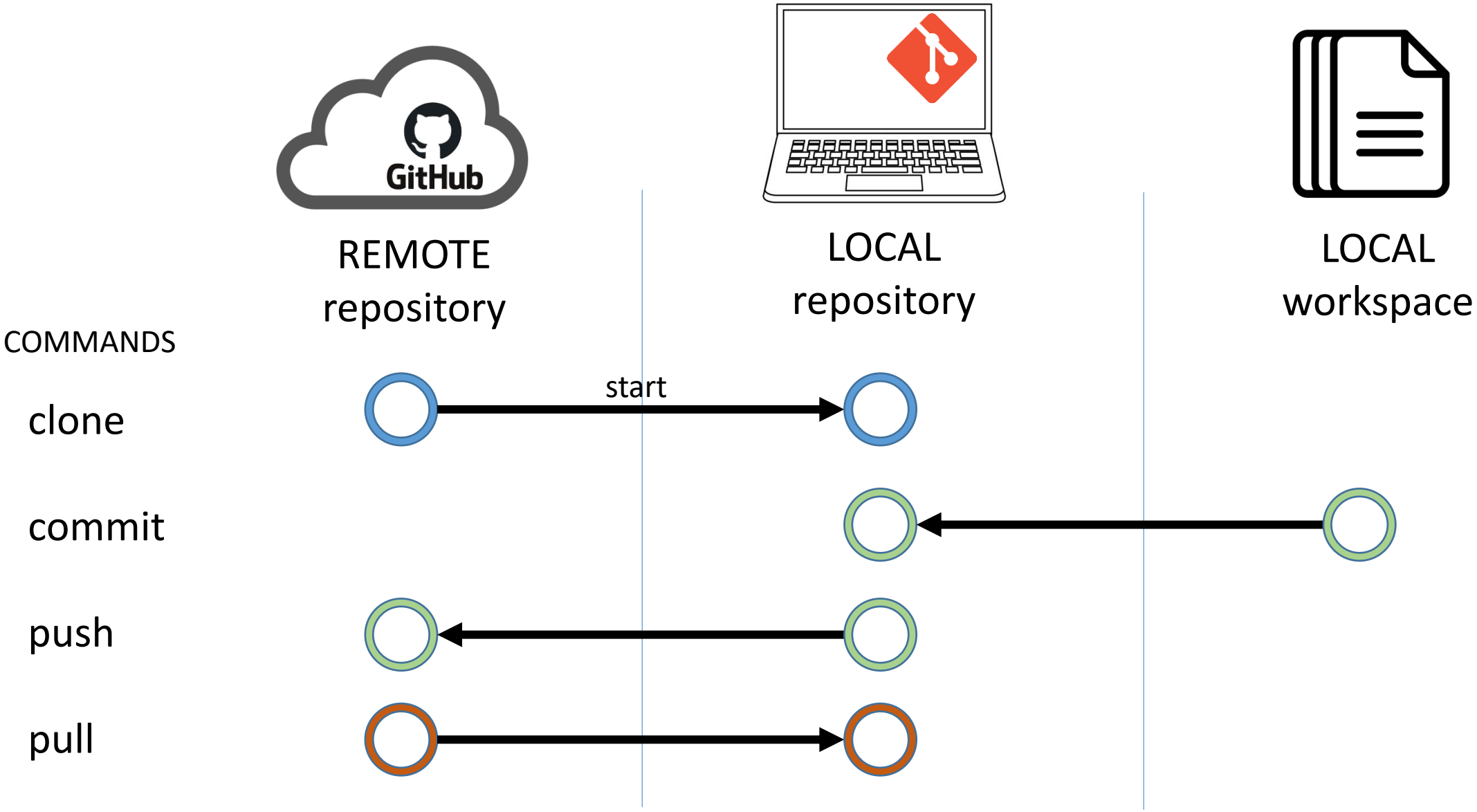
COMMANDS

add

commit



Git + GitHub workflow




In case of fire
1. git commit
2. git push
3. leave building

GitHub

REMOTE repository

lib.py

```
def div(a, b):
  return a/b
```

* NOTE: we are ignoring branches for now

GitHub

REMOTE repository

lib.py

```
def div(a, b):
  return a/b
```

clone

Alice

LOCAL repo

lib.py

```
def div(a, b):
  return a/b
```

* NOTE: we are ignoring branches for now

GitHub

REMOTE repository

lib.py

```
def div(a, b):
  return a/b
```

clone

Alice

LOCAL repo

lib.py

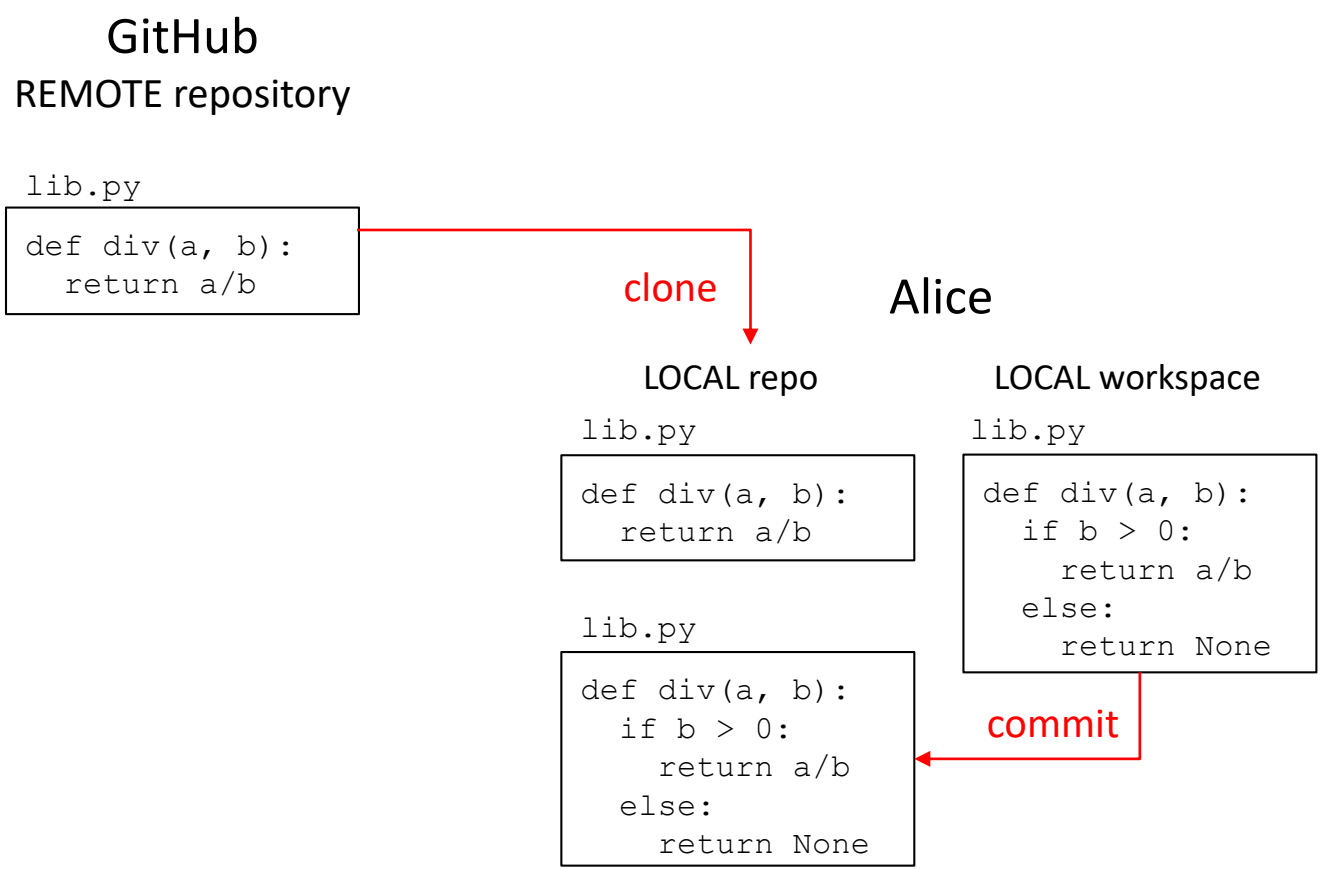
```
def div(a, b):
  return a/b
```

LOCAL workspace

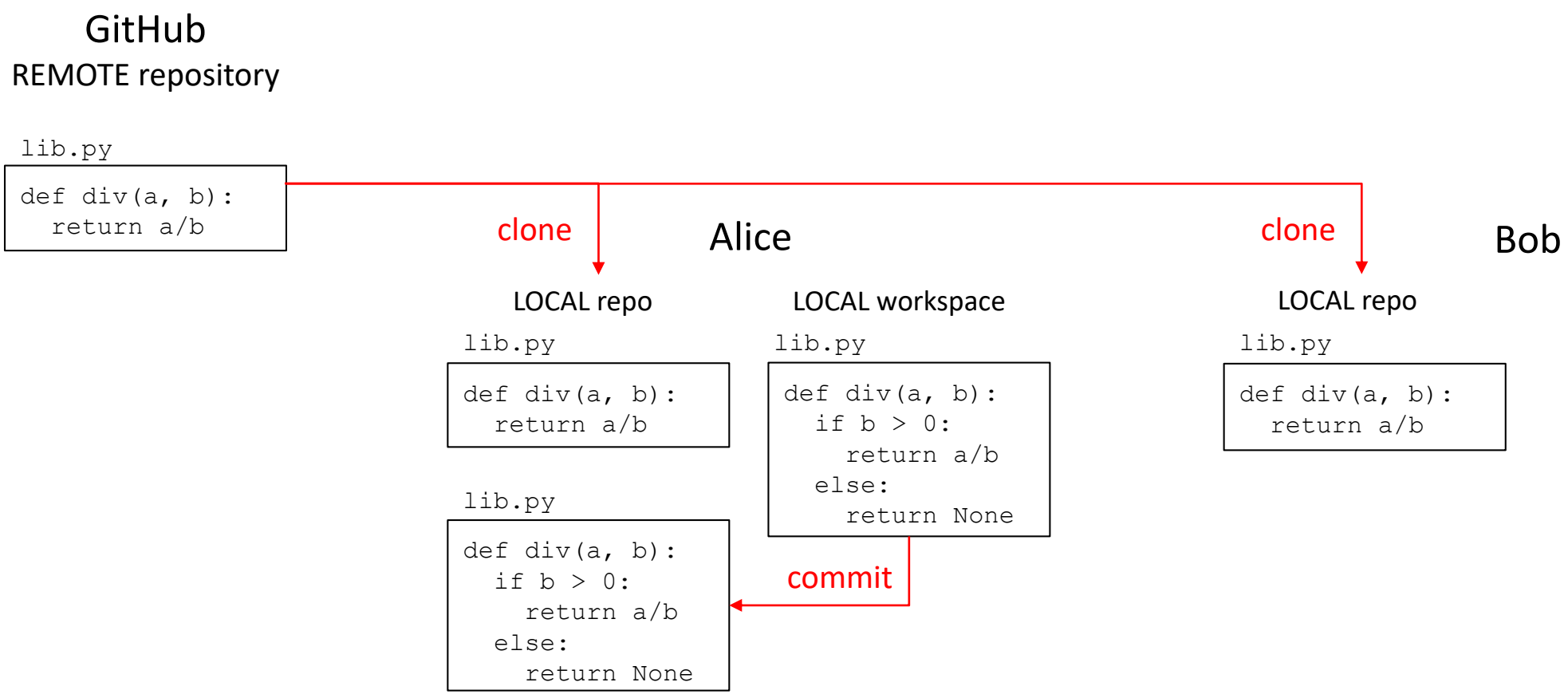
lib.py

```
def div(a, b):
  if b > 0:
    return a/b
  else:
    return None
```

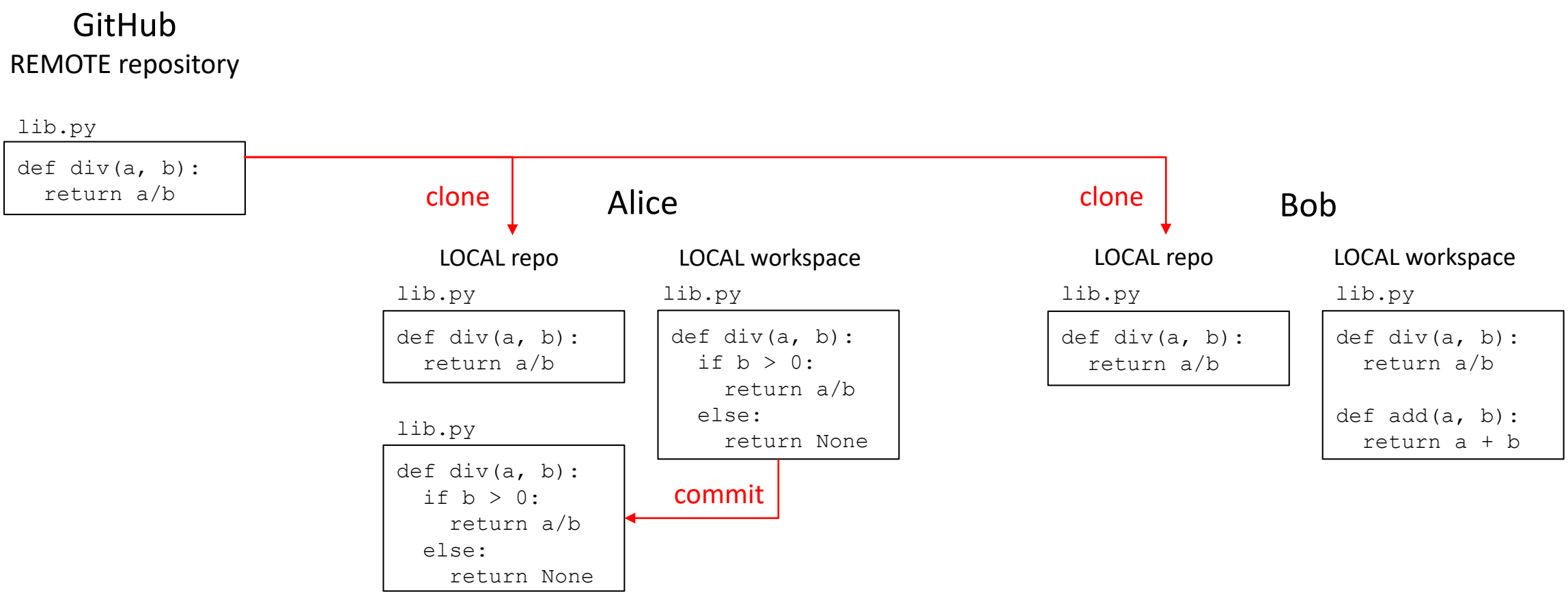
* NOTE: we are ignoring branches for now



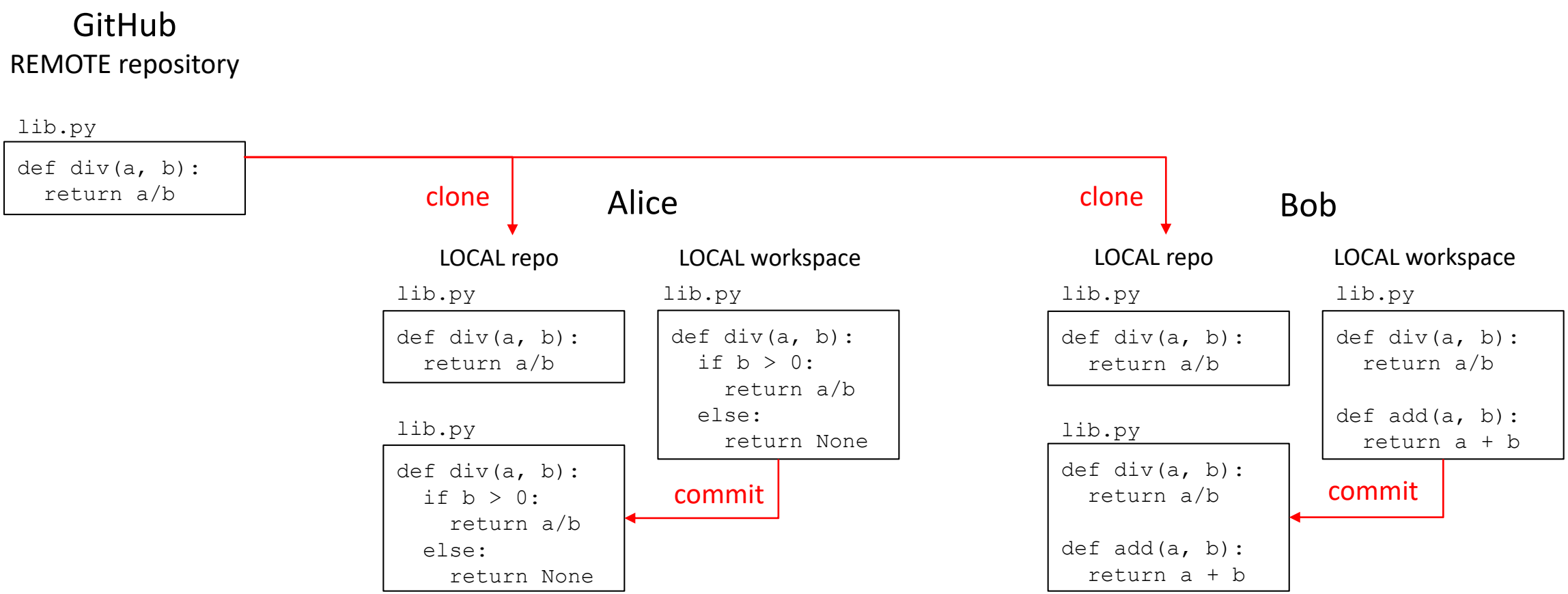
* NOTE: we are ignoring branches for now



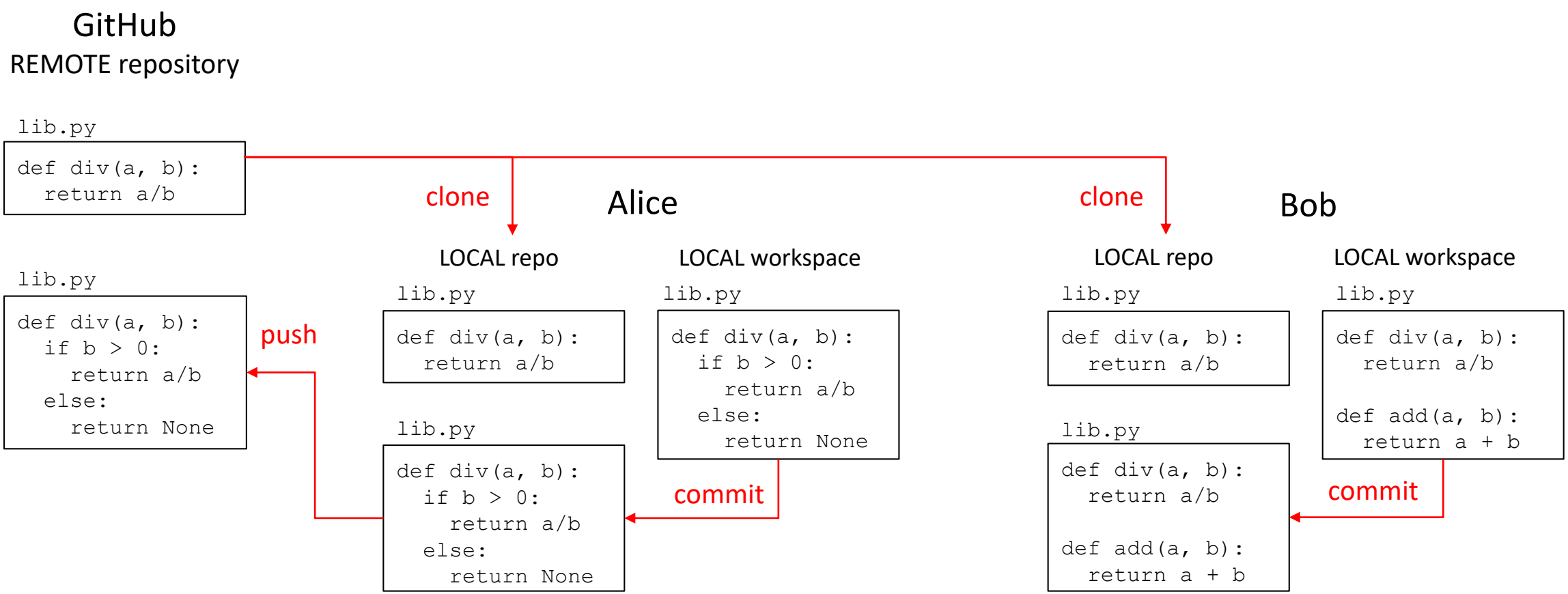
* NOTE: we are ignoring branches for now



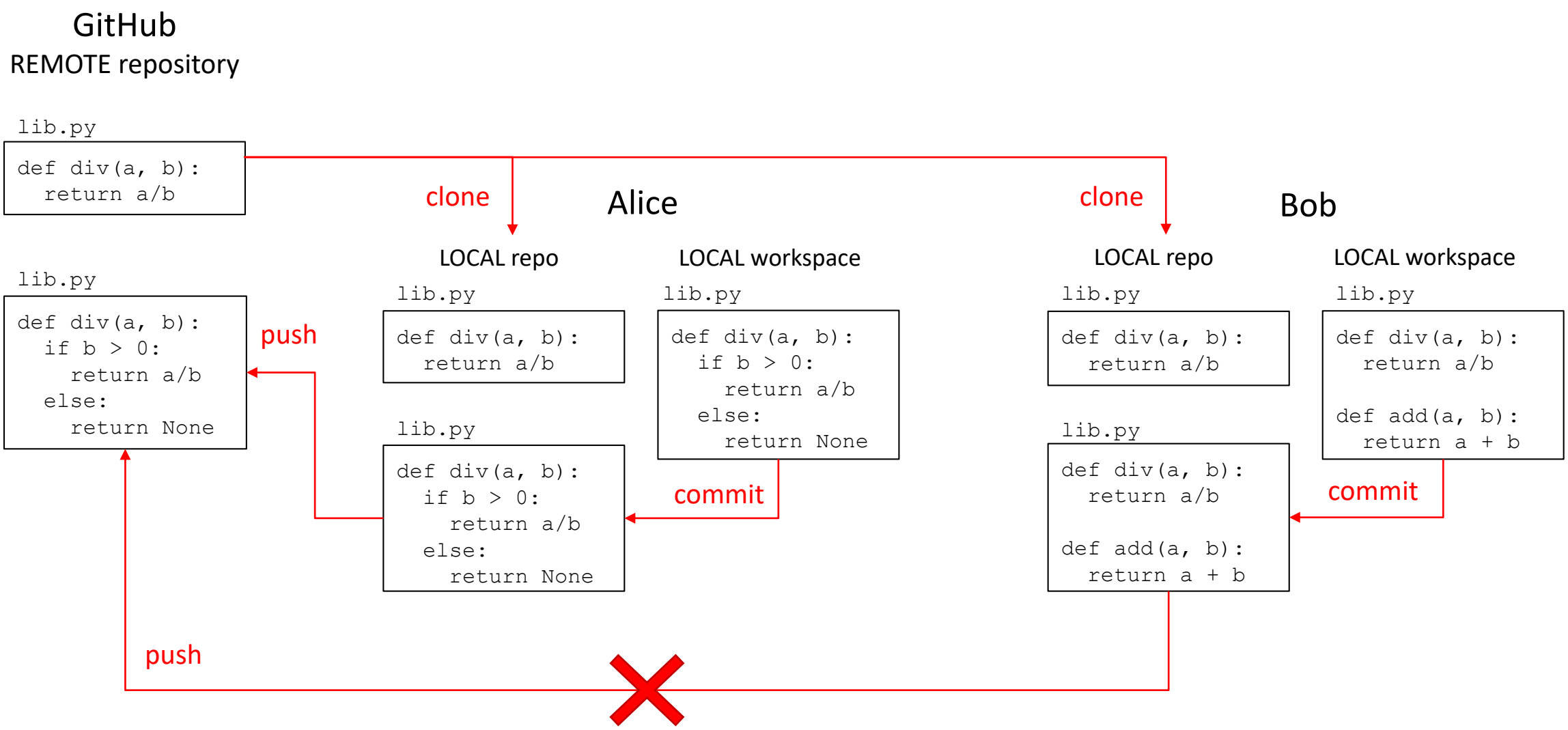
* NOTE: we are ignoring branches for now



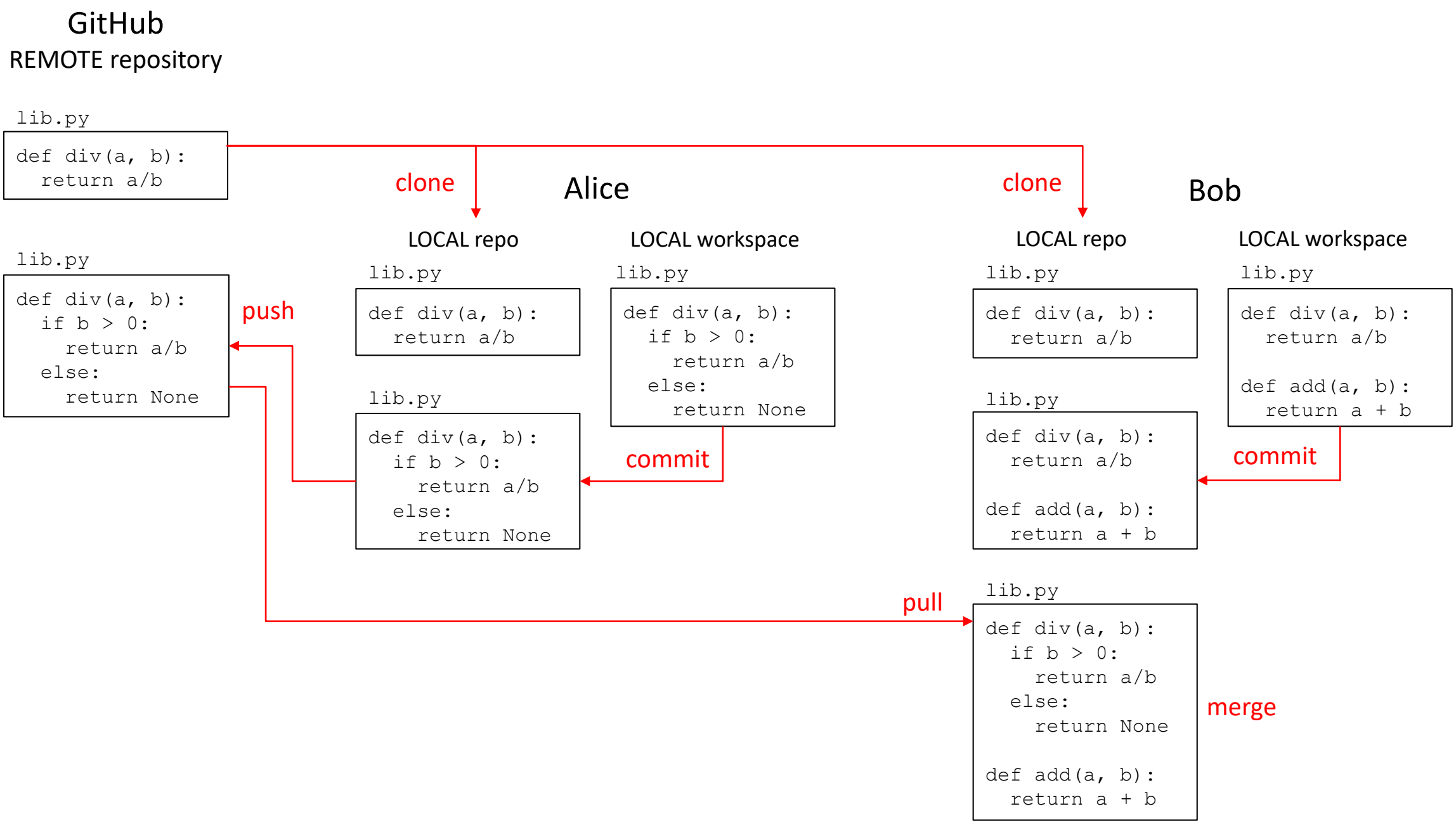
* NOTE: we are ignoring branches for now



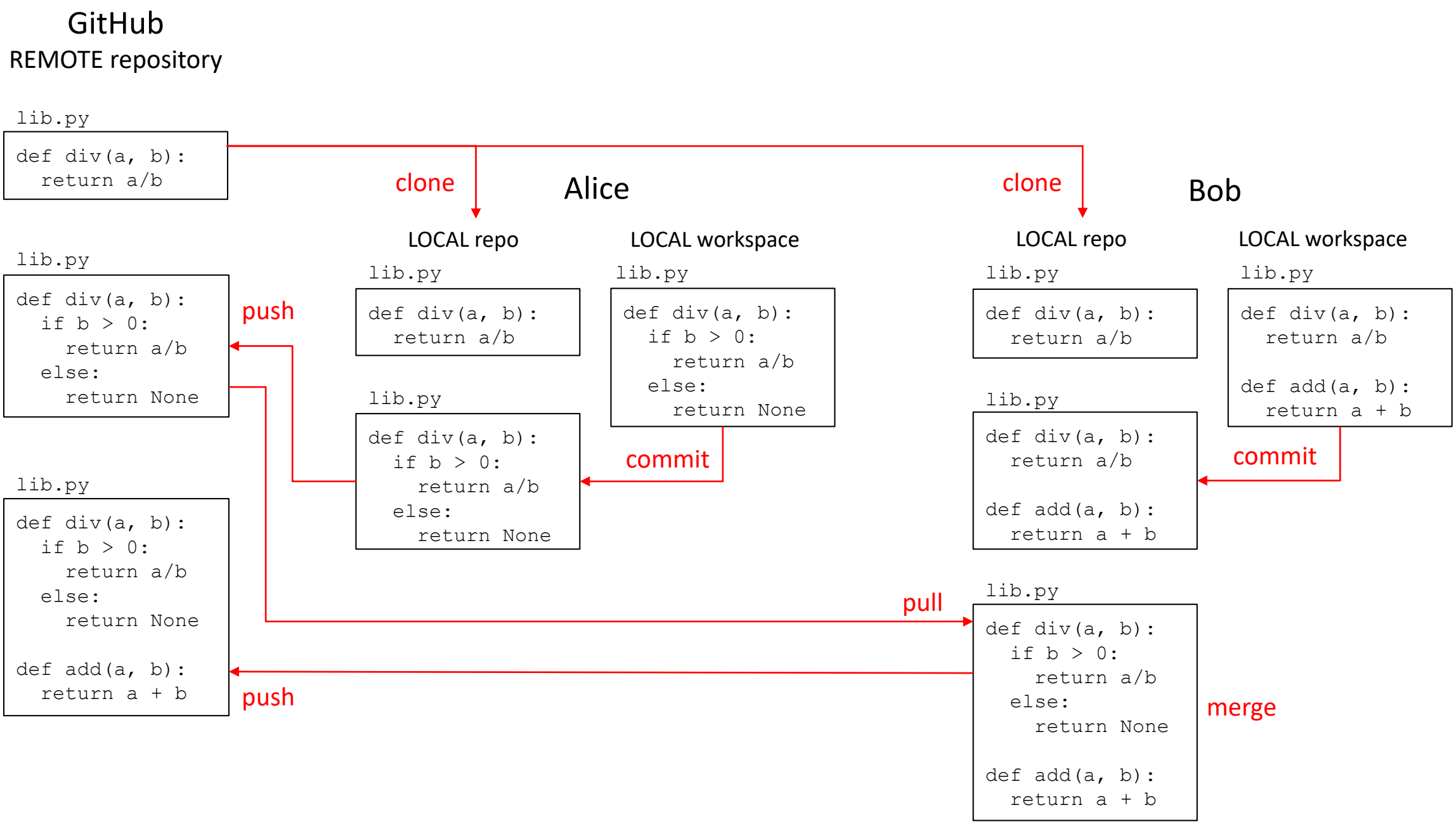
* NOTE: we are ignoring branches for now



* NOTE: we are ignoring branches for now

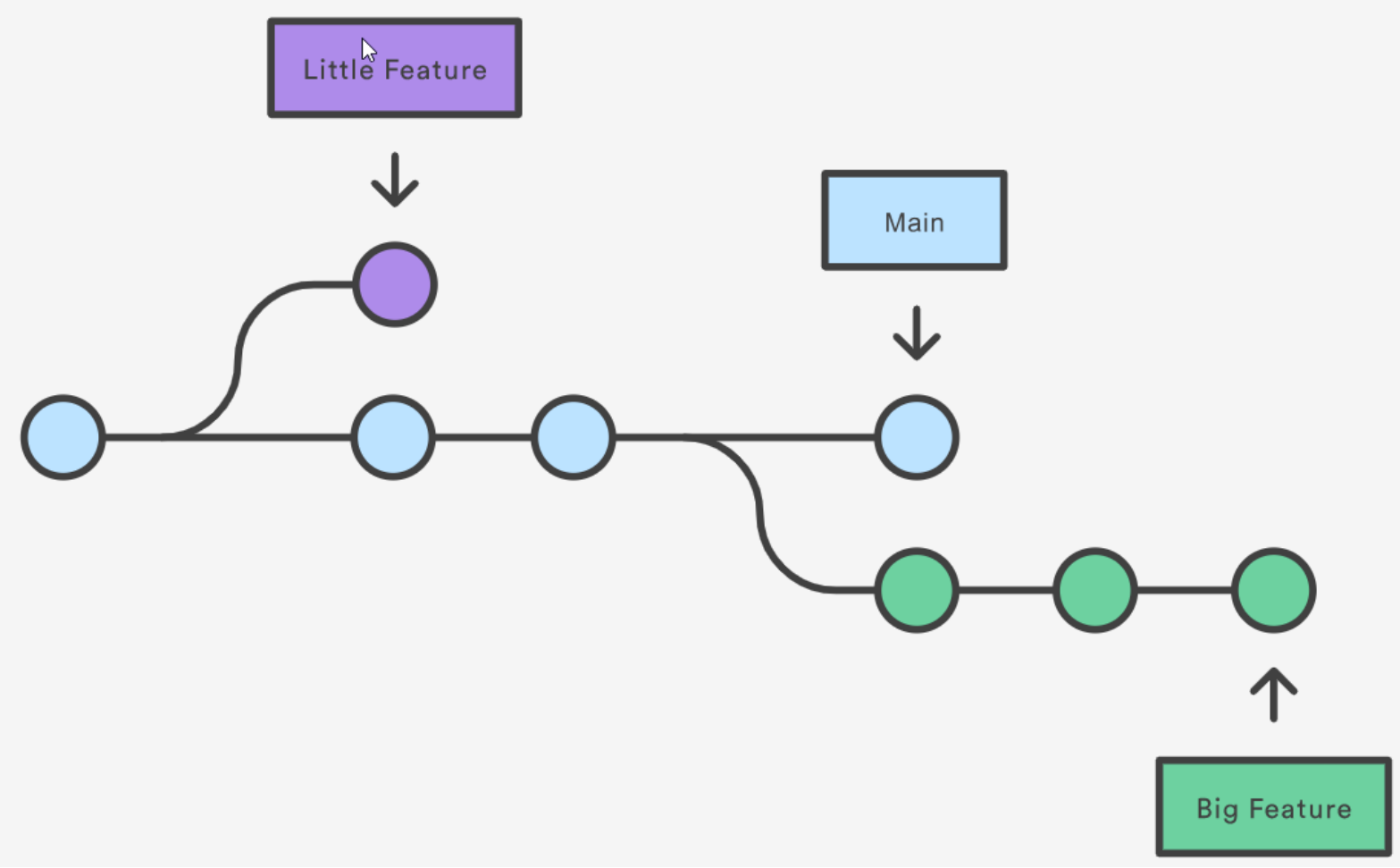


* NOTE: we are ignoring branches for now



* NOTE: we are ignoring branches so far

Git branching



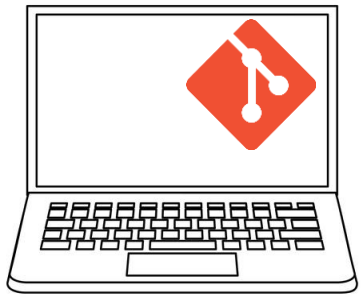
Commit on “Little Feature” branch

Commit on “Main” branch

Commit on “Big Feature” branch

Time

Git branching



REMOTE repository

LOCAL repository

LOCAL workspace

COMMANDS

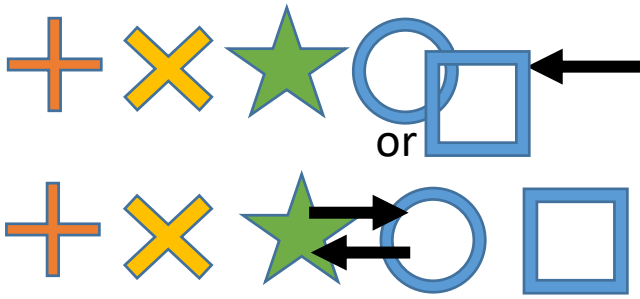
branch

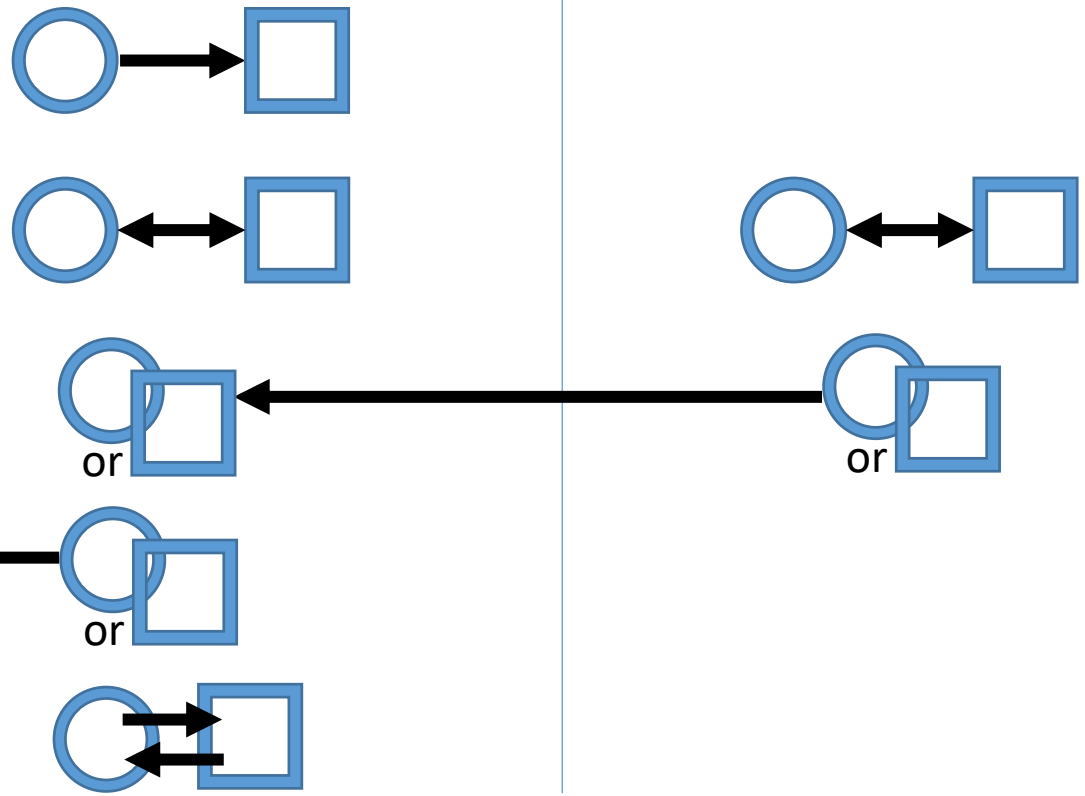
checkout

commit

push

merge



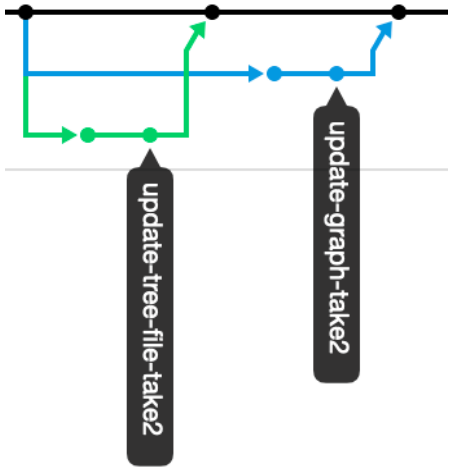


Git branch workflow

1. Check out the `main` branch (do NOT create new branches from non-`main` branches)
2. Create and checkout a new branch.
3. Run `git push –set-upstream origin <newbranchname>` to track the branch in your remote repo
4. Edit your files and make all the commits as usual.
5. Run `git push –set-upstream origin <newbranchname>` to upload the commits to the branch on remote
6. Go to the repo on GitHub and create a “pull request” to start to merge your dev branch to main.
7. Fix any merge conflicts if they exist.
8. Back on your computer, check out the `main` branch and run `git pull`, to copy the changes in the remote repo to your local repo.

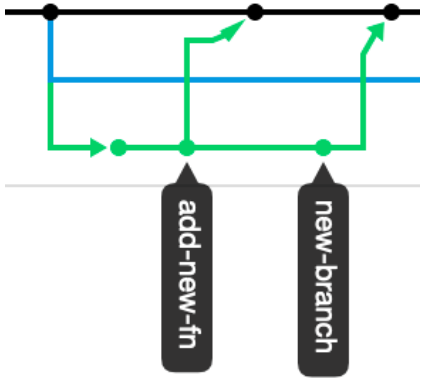
Git branching

From your GitHub repo page going to `Insights` → `Networks` will show a graphical representation of your repo. The example at right shows the `main` branch in black and two dev branches (green and blue), both of which have been merged back into the `main`.

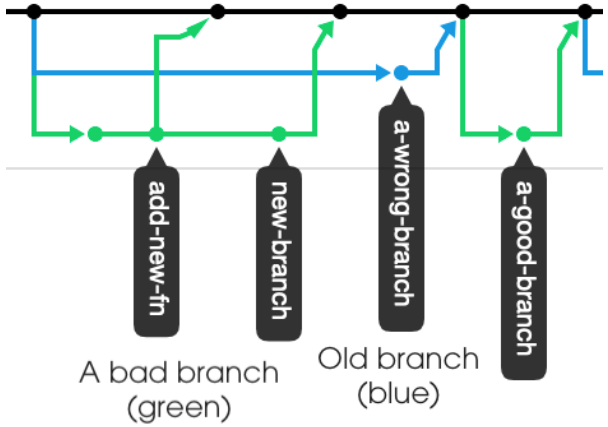


Examples of bad branching:

Creating a branch from another dev branch (instead of from main)



Creating a new branch from an older commit on main.



Markdown syntax

```
# Header 1
## Header 2
### Header 3
…
```

Header 1

Header 2

Header 3

Monospace code block:

```
def my_func(x):
    y = x + 3
    return y
```

Bold text
__Bold text__

Italic text
Italic text

Horizontal lines:

> Block quote
>> More indenting
>>> And even more indenting

Enumerate and itemize

1. Pie
2. Cake
3. Flan

- Broccoli
- Carrot
- Lettuce