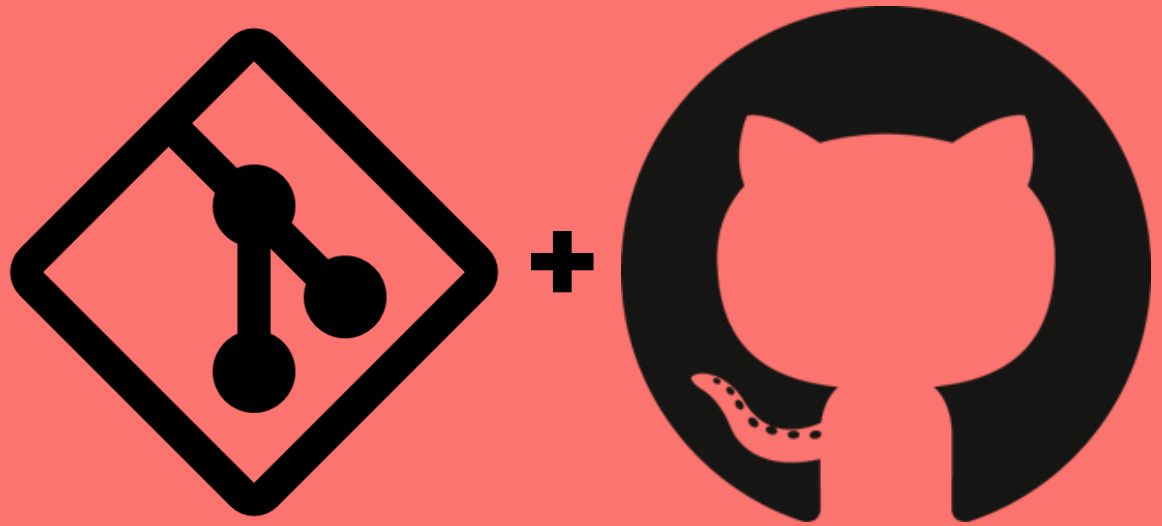
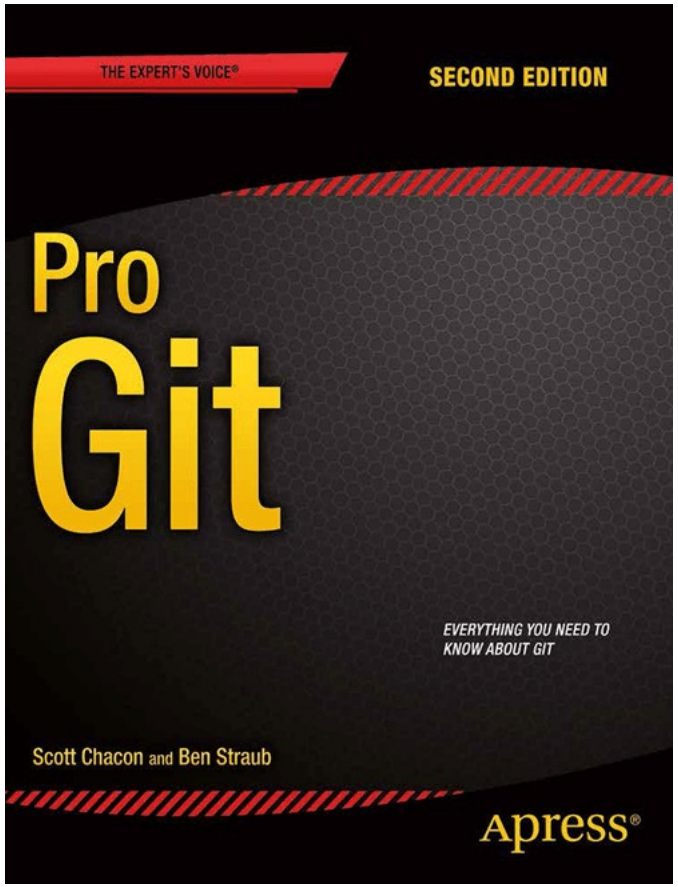
Git

In Practice



1. Git Fundamentals
2. Undo working tree and staging area

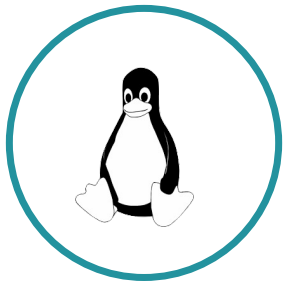
ProGit Book



https://git-scm.com/book/en/v2

Git – A Brief History

1992 – 2001



Linux

Software changes to the Linux Kernel were passed around as patches

2002 – 2005

BitKeeper

Linux kernel project began using a proprietary DVCs – BitKeeper

2005

Git

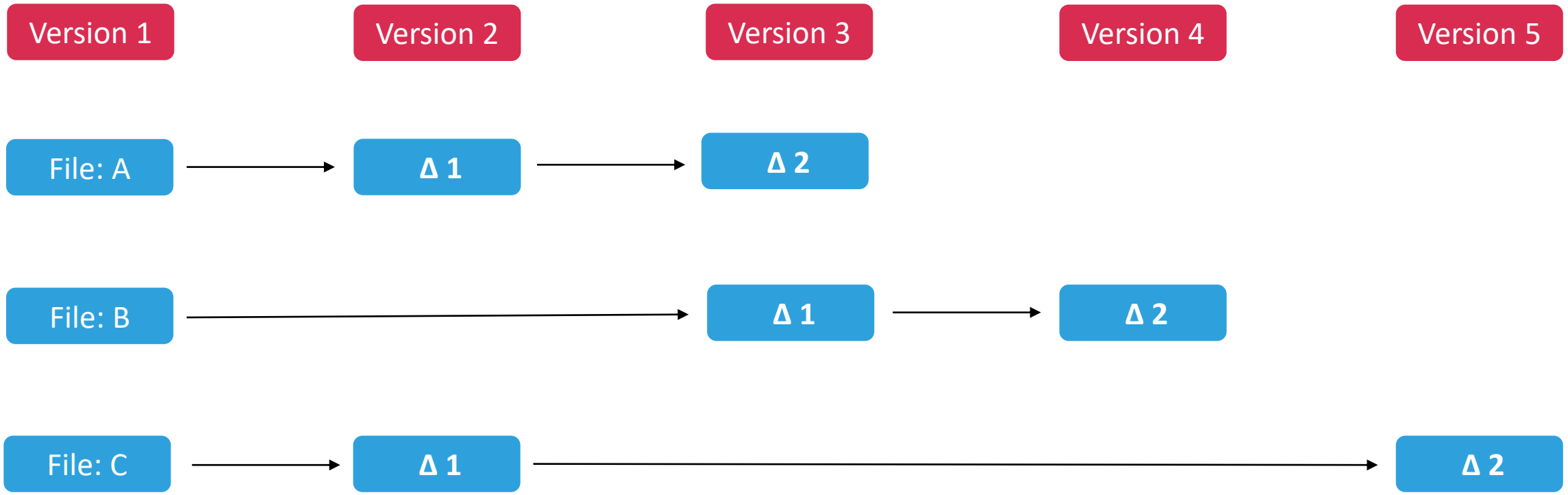
Linus Torvalds writes Git to replace BitKeeper

Goals of Git

- ❑ Fully distributed
- ❑ Simple design which handles large projects efficiently – speed and data size
- ❑ Strong support for non-linear development (thousands of parallel branches)

A Comparison

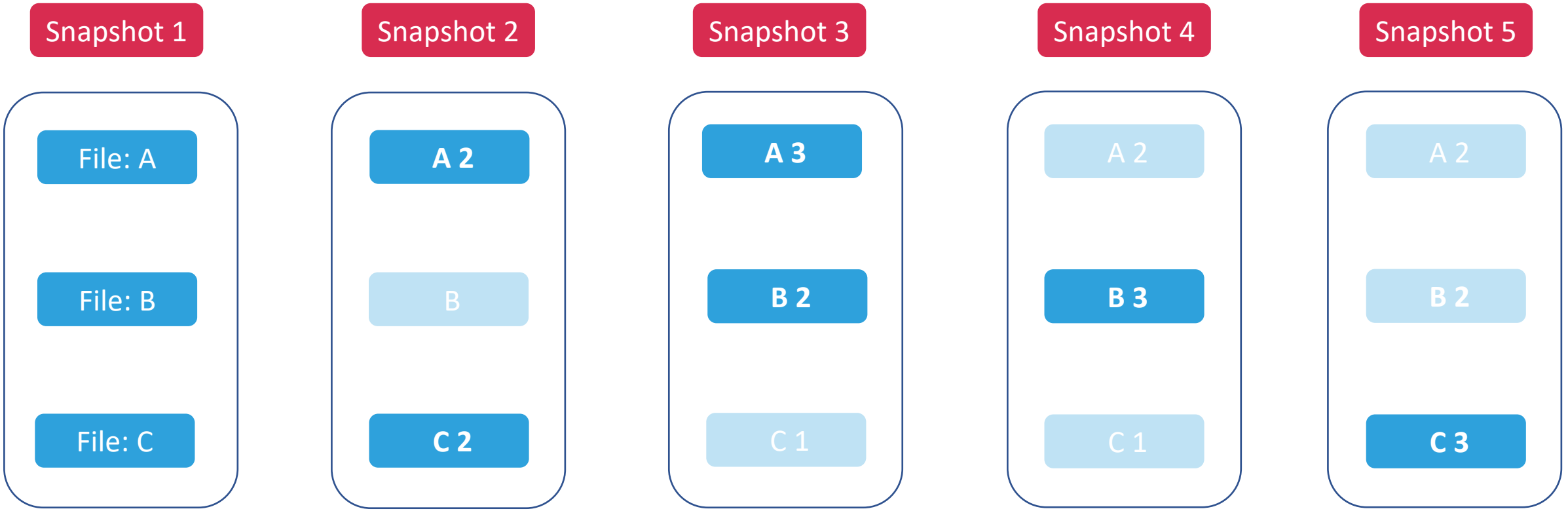
Other Version Control Systems



Delta-based Version Control

A Comparison

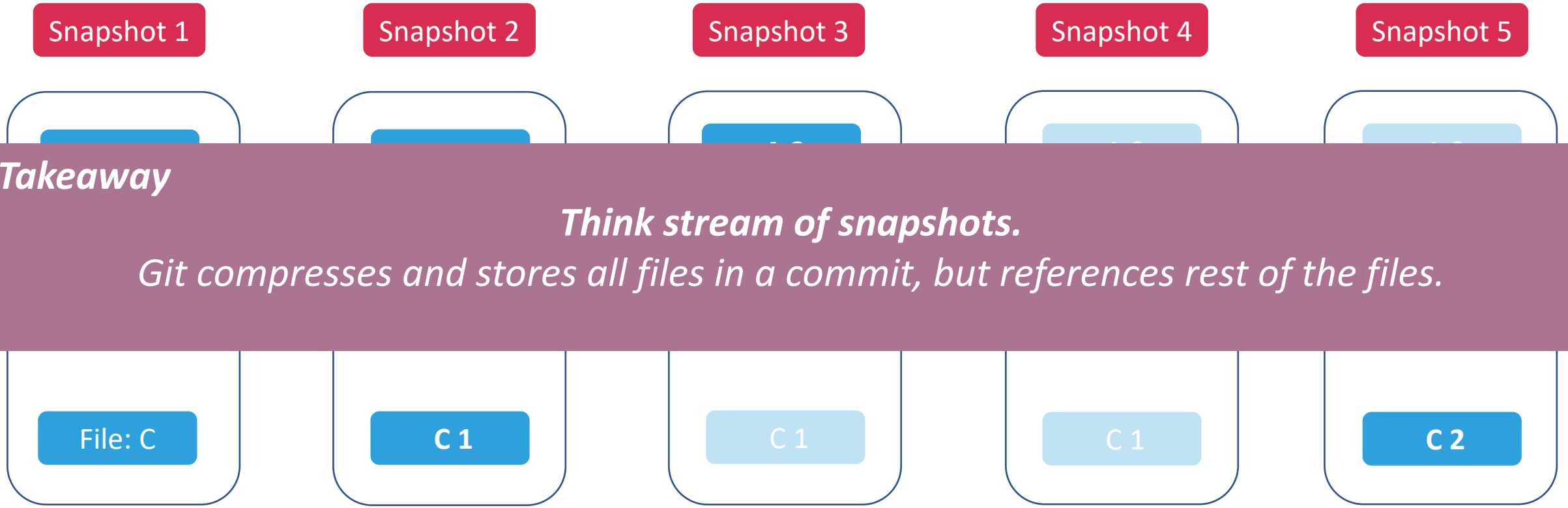
Git



Stream of **Snapshots**

A Comparison

Git



Takeaway

Think stream of snapshots.

Git compresses and stores all files in a commit, but references rest of the files.

Stream of **Snapshots**

Content–based Addresses

- Git uses a “content-based addressing” naming scheme
- The name of an object in content-based addressing is derived from its content
- The idea of content-based addresses are:
  - Names *succinctly* summarize the content – names should be much shorter than the contents
  - Different contents get unique names
  - The same content always gets the same name

Git Hash

- Git uses SHA–256 (SHA-1 prior, 2020) to generate a 40-char hash

```
3c2571e28d1269ed545572262084c13176271ce2
```

```
$ git hash-object index.html
```

- Checksums are used to validate the integrity of the data
  - Modifying the data in any way changes the hash value

Git Installation

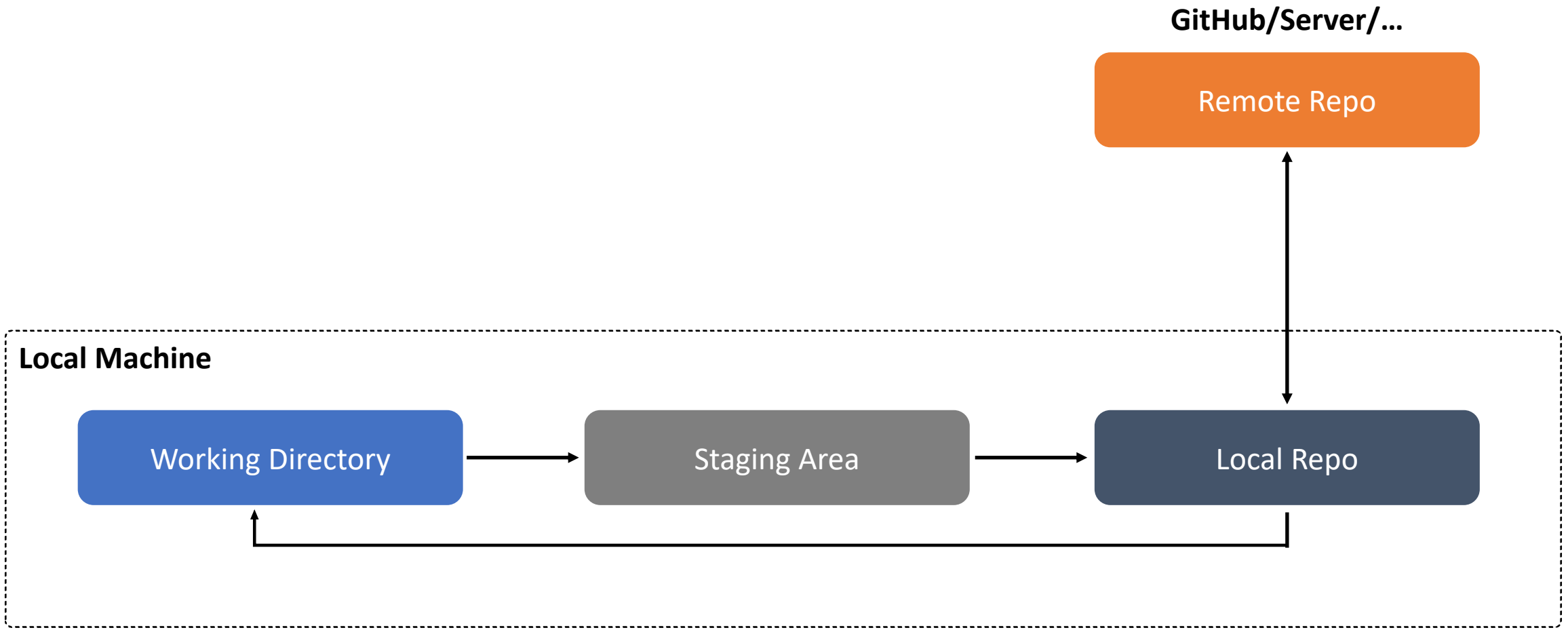
Official – git-scm

https://git-scm.com/downloads

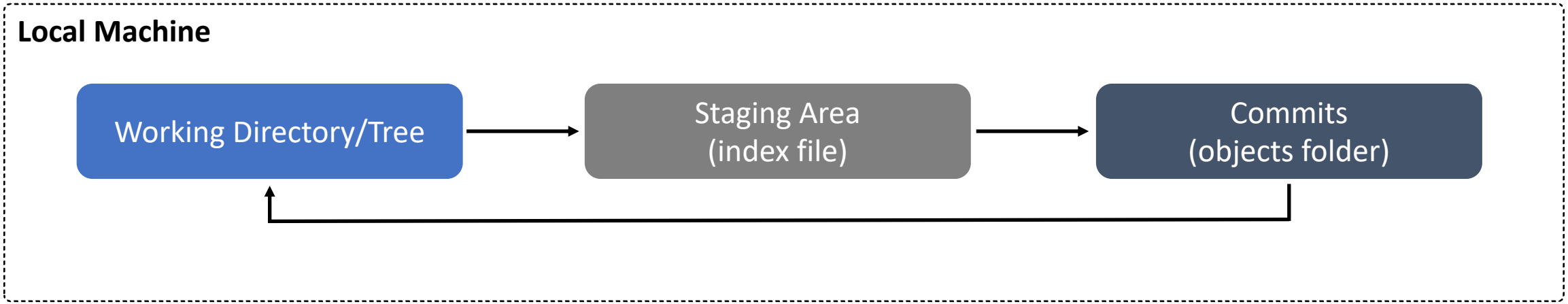
Github

https://github.com/git-guides/install-git

A Distributed Version Control System



Git Project Components

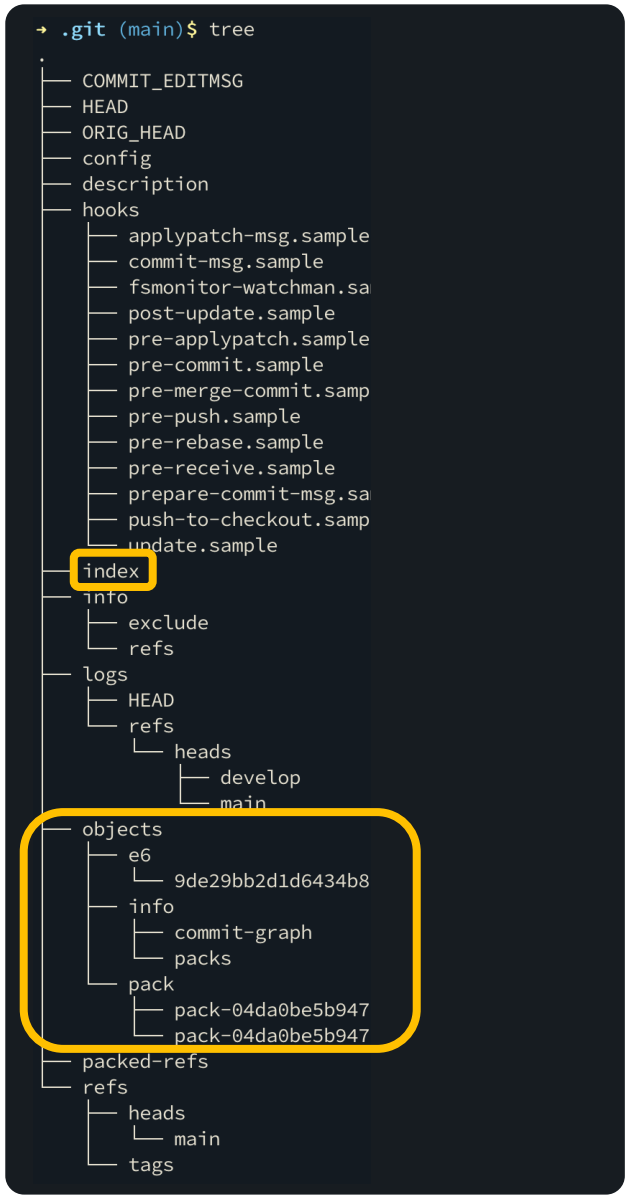



```
$ git init
```

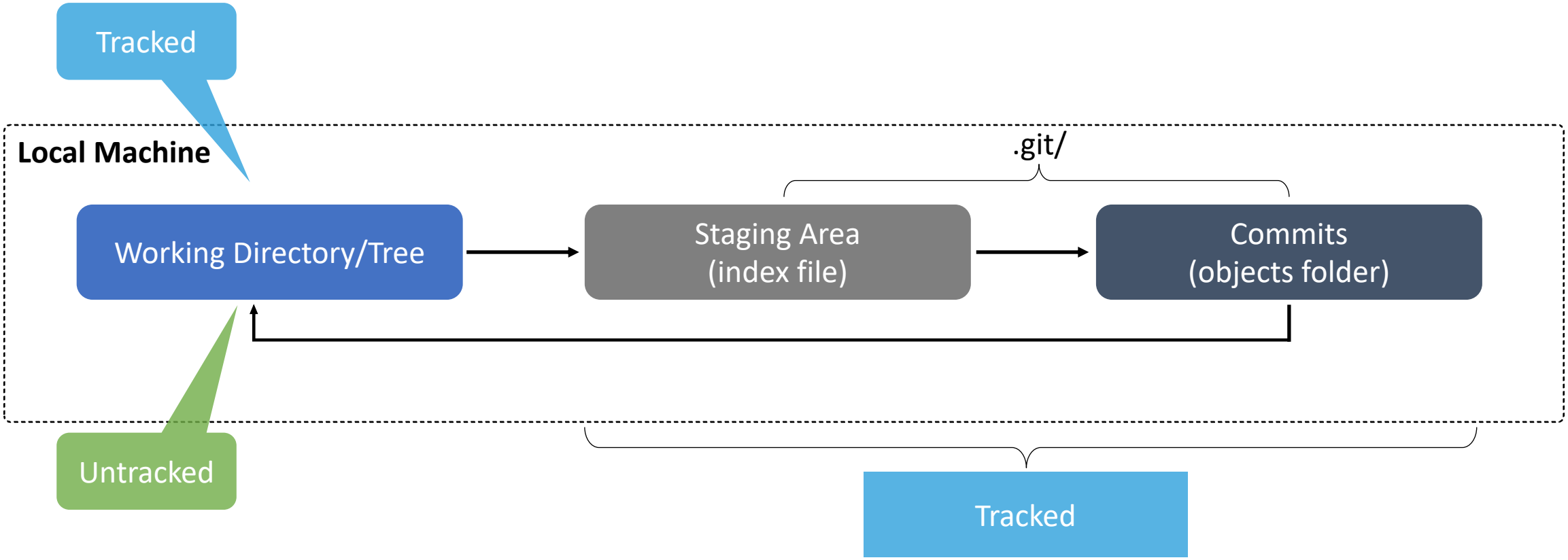
.git/

Staging file

Commits DB



Git Project Components




```
$ git ls-files
```

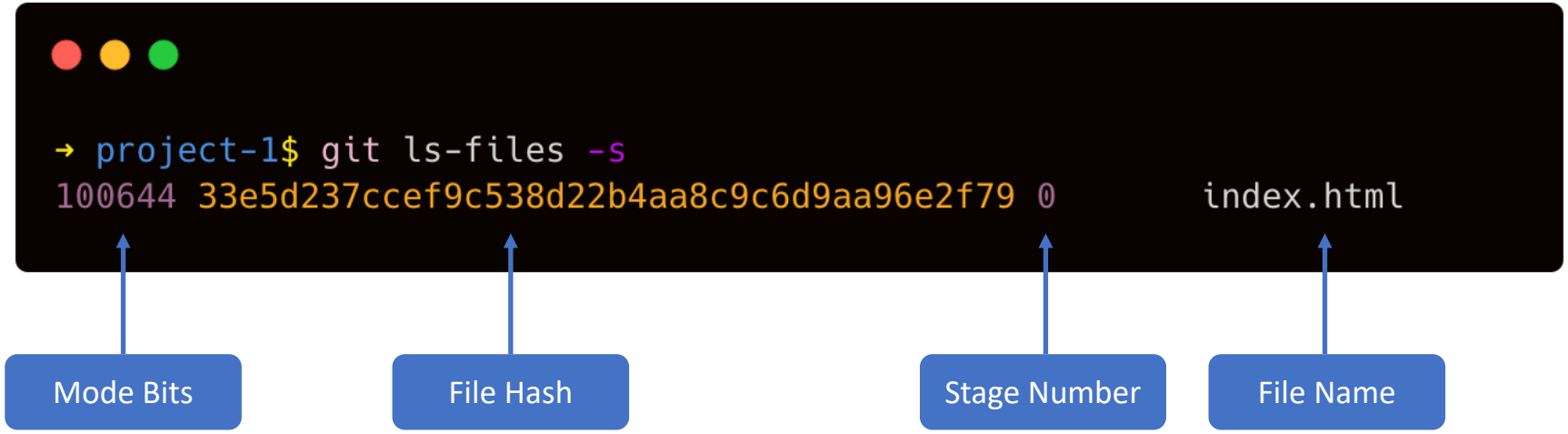
```
$ git log
```

git status

- The `git status` command displays the state of the working tree and the staging area
- It shows what has been going on with `git add` and `git commit`
- It displays the
  - *Tracked files* that have differences between the working tree and the index file
  - *Untracked files* in the working tree that are not tracked by git
- It does not display the status of the files that are ignored (.gitignore)

git ls-files

- `git ls-files` is a debug utility for inspecting the state of the Staging Index tree
- `git ls-files` with a `-s` or `--stage` displays additional metadata for the files in the Staging Index




```
$ git hash-object index.html
```

```
33e5d237ccef9c538d22b4aa8c9c6d9aa96e2f79
```

git log – TODO

- The git log command displays committed snapshots
-
- `git ls-files` with a `-s` or `--stage` displays additional metadata for the files in the Staging Index

Git States

Untracked (New) – A new untracked change (file) exists in the working directory

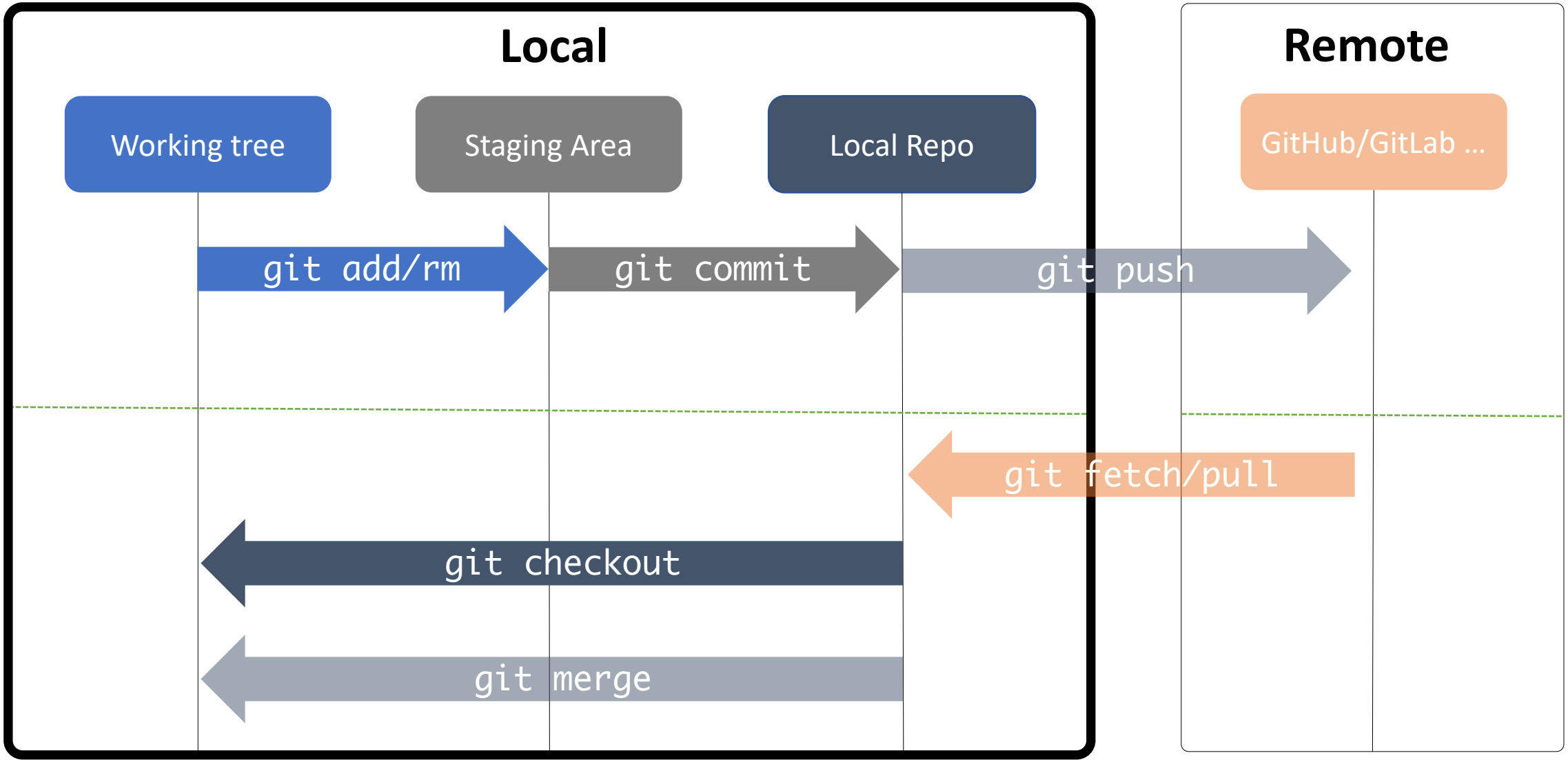
Modified – A tracked file is modified in your working directory

Staged – Changes are tracked and ready to be committed

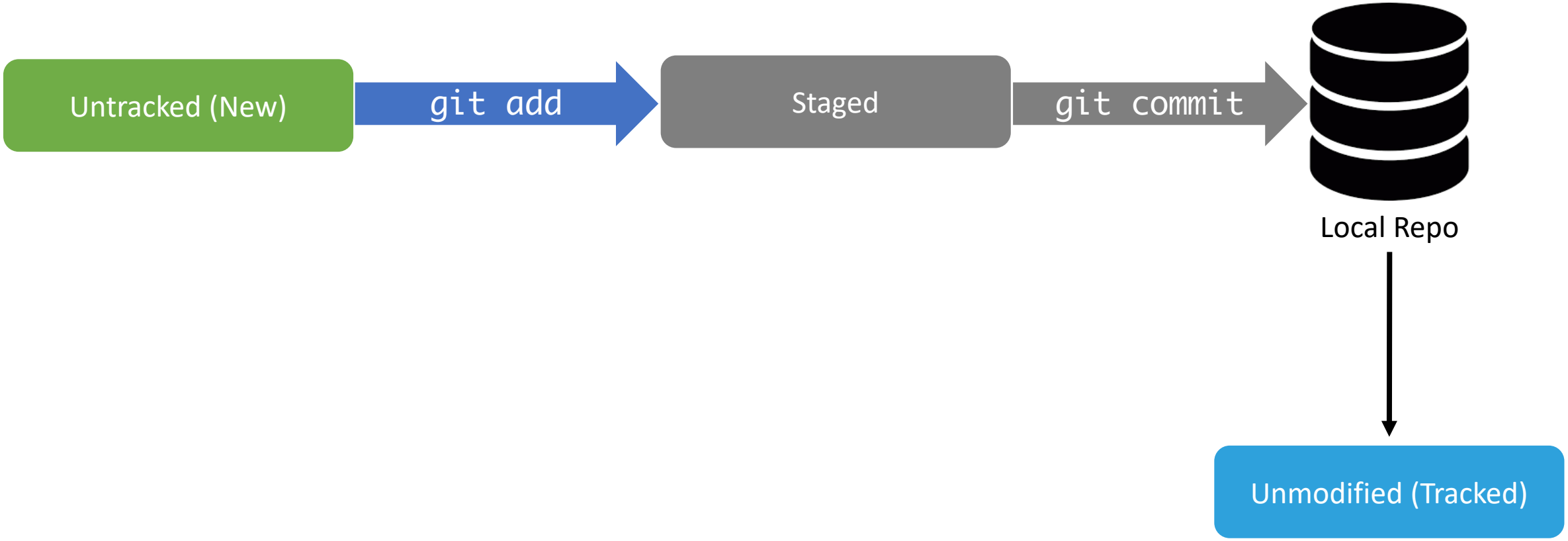
Local Repo – Changes are committed and stored in the Object Database

Unmodified (Tracked) – The working directory is clean – nothing to commit

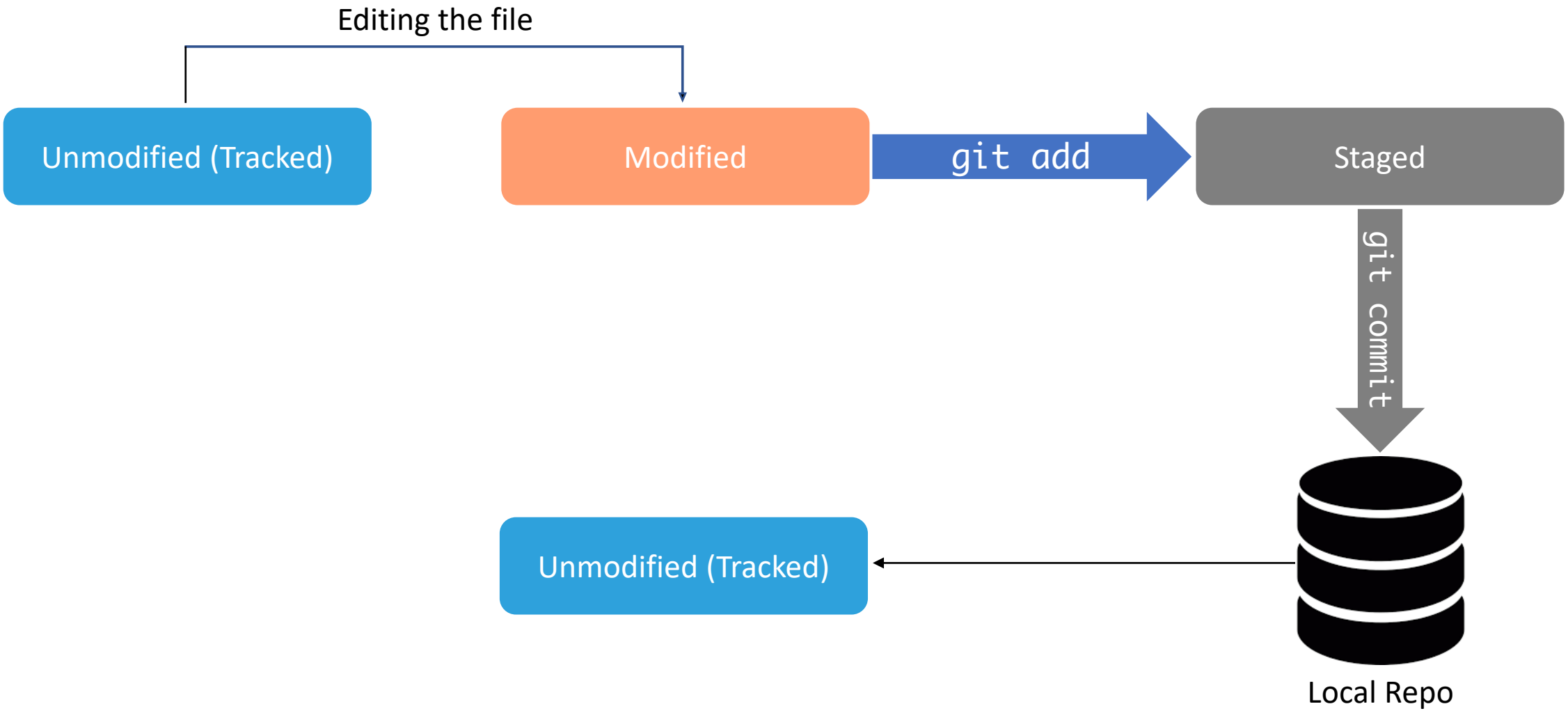
Project Components – Commands



Scenario 1 – Add Changes



Scenario 2 – Edit



Let us practice

Commits And Data Integrity

- Git checksums include meta data about the *commit* including
  - Commit message
  - Committer
  - Commit date
  - Author
  - Author create date
  - Working Directory/tree hash
  - The previous *commit's* (parent) hash

Commits And Data Integrity

- Git assures the integrity of the stored data by using checksums as identifiers

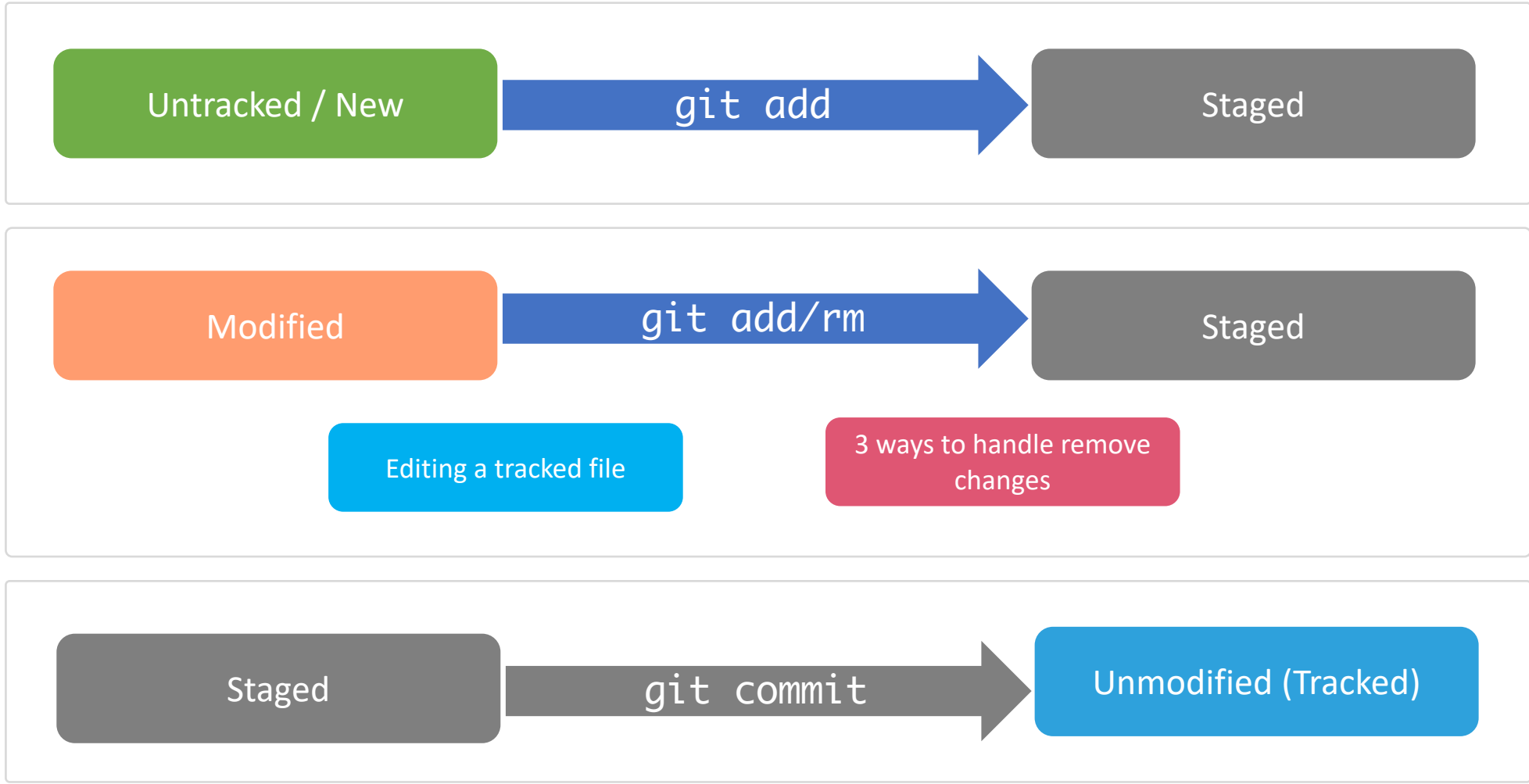
- This way, Git also ensures historical chain of commits cannot be edited

Scenario 3 – Remove Changes

Action	Command	Result
To delete a tracked file	`git rm` *`filename`*	Removes files from working directory and adds a delete marker to staging area
If a tracked file was deleted from the working directory	`git rm` *`filename`* OR `git add` *`filename`*	Adds a delete marker to the staging area
To remove the file from the staging area	`git rm --cached filename`	Removes the file from the staging area

Let us practice

Scenarios – Summary



Takeaways

- ✓ Stream **of snapshots**
  - Git doesn’t store data as a series of changesets or differences
  - Instead, it stores data as a series of *snapshots*
  - You can consider each commit as a “mini file system”

- ✓Content-based addressing
  - Git uses a “content-based addressing” naming scheme
  - The name of an object in content-based addressing is derived from its content

Takeaways

- ✓ Three Project Components
  - Working Tree
  - Staging Area
  - Commits History
    - Git uses an object database to store the commits

Takeaways

✓ Commands

Start a working area

1. `git init`

Work on the current change

2. `git add`
3. `git rm`
4. `git rm --cached`
5. `git checkout`

Grow, mark and tweak your common history

8. `git commit`

Show current status and commit history

10. `git status`
11. `git ls-files`
12. `git log`