Git: Day 2

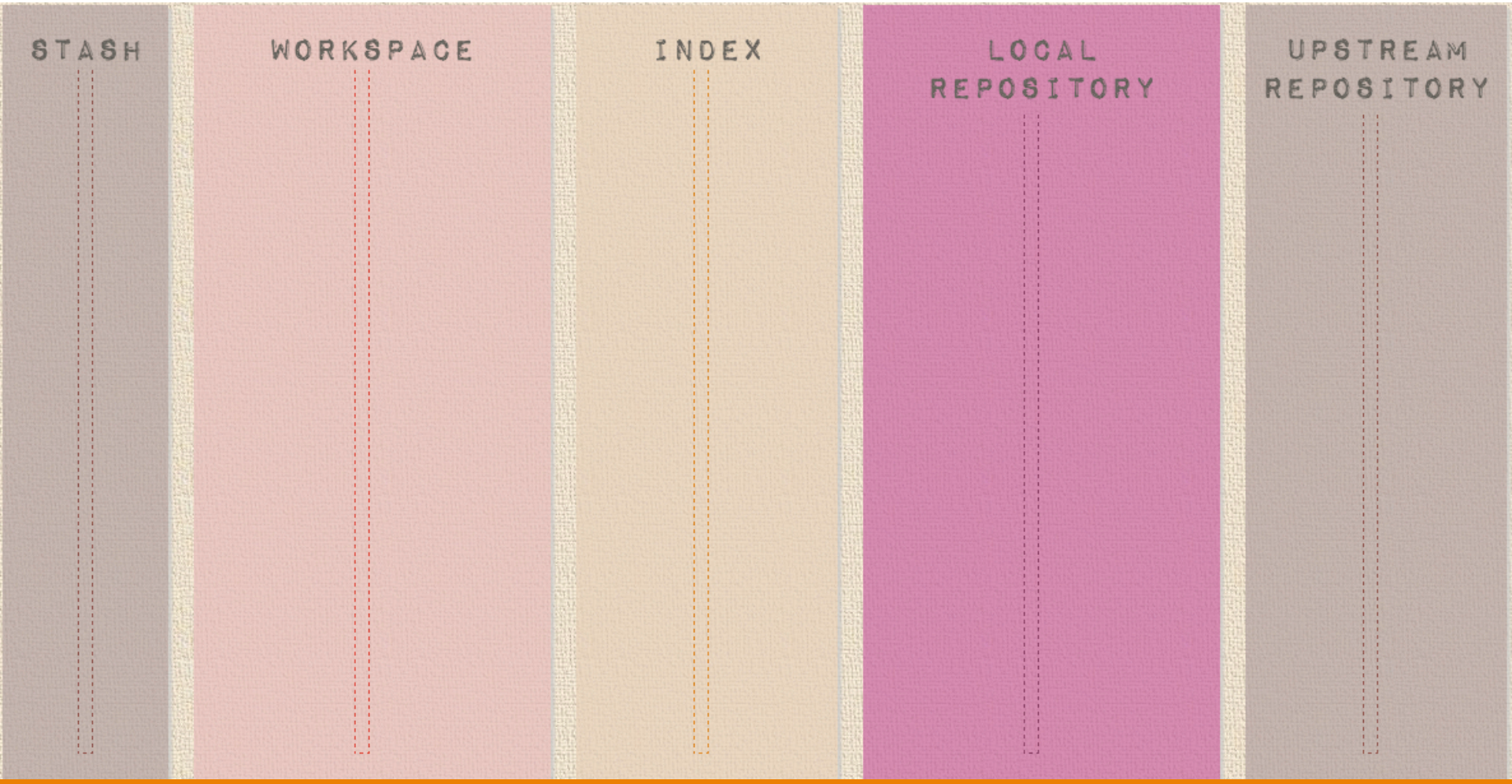
First of all…

Clear your mind

GIT BASICS

Internal Areas

Git has the following areas:



Working Directory / Workspace

The filesystem where your files are managed!

Local repository

The Git repo in your local machine.

You can create that repo with git init or git clone commands.

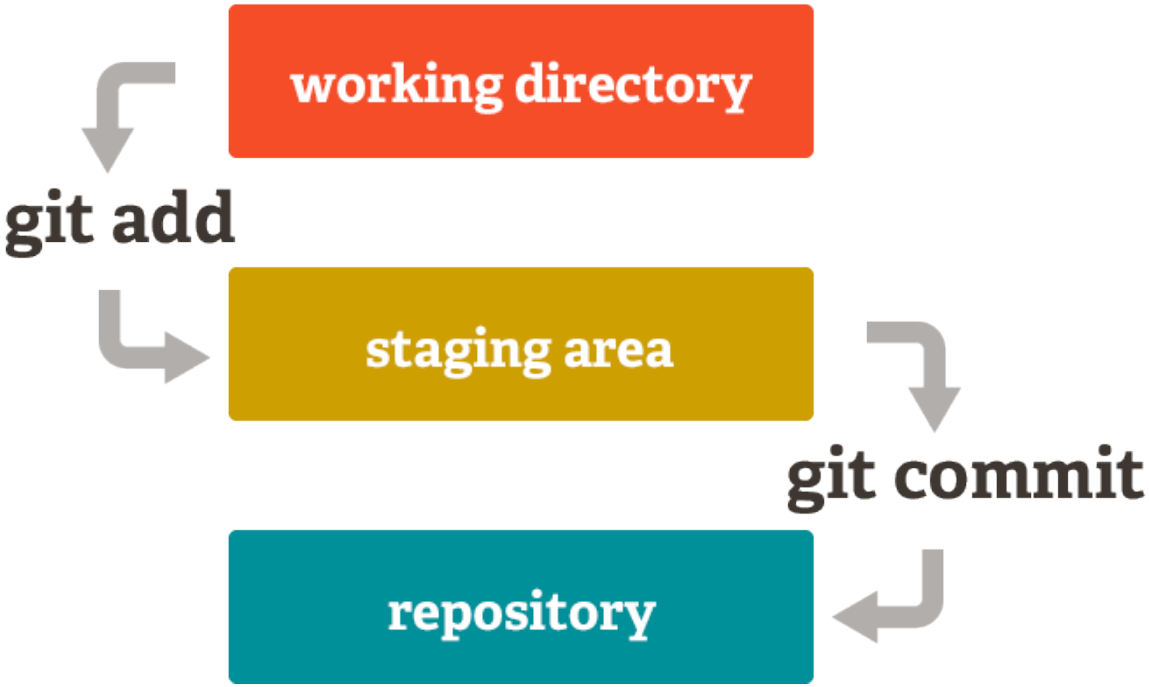
Upstream repository

The remote repository where you can push or pull your commits.

Default name is ‘origin’

Staging Area / Index

Intermediate zone where next commit is prepared.



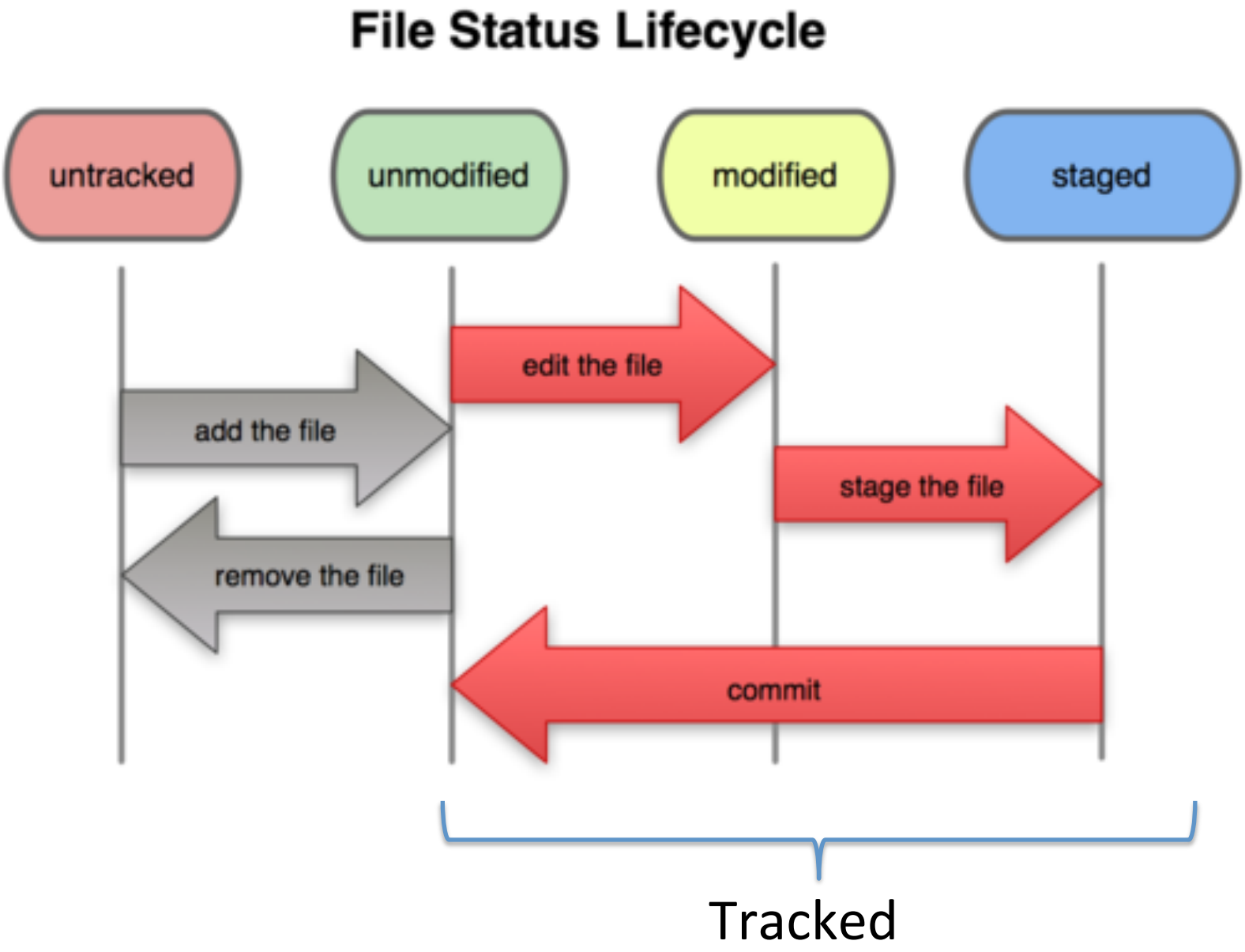
File Status Lifecycle

Each file in your working directory can be in one of two states: tracked or untracked.

Tracked files are files that were in the last snapshot; they can be unmodified, modified, or staged.

Untracked files are everything else.

File Status Lifecycle



Status

You can check your repo status:

```
$ git status
```

GIT INTERNALS

git HEAD

HEAD is a pointer that points to the current branch.

When you change to a different branch, HEAD changes to point to the new one.

The current HEAD is **local to each repository**, and is therefore **individual for each developer.**

Hands-on

```
$ cd .git
$ cat HEAD
        ref: refs/heads/master

$ git checkout –b newFeature
$ cat HEAD
        ref: refs/heads/newFeature
```

Double Dash --

You can use -- to:

- Separate options from a list of arguments:

```
$ git diff –w master origin -- tools/Makefile
```

- Separate an explicitly identify filenames

```
# Checkout the tag named “main.c”
$ git checkout main.c

#Checkout the file named “main.c”
$ git checkout -- main.c
```

Basic Flow: First Commit

1. Adding a File to Your Repository

- One file

```
$ git add <filename>

$ git add *.c
$ git add index.html
```

- A directory

```
$ git add <directory>
```

Basic Flow: First Commit

2. Commit One file

```
$ git commit –m “message”
```

If the file is managed by the repo:

```
$ git commit –am “message”
```

3. Remove a file from the staging area

#Before first commit

$ git rm --cached <file>

#After first commit

$ git reset HEAD <files>

Basic Flow: First Commit

REFS AND SYMREFS

git Ref

- A ref is a SHA1 hash ID that refers to an object within the Git object store. Although a ref may refer to any Git object, it usually refers to a commit object.

- A symbolic reference , or symref , is a name that indirectly points to a Git object. It is still just a ref.

- Local topic branch names, remote tracking branch names, and tag names are all refs.

Symrefs

HEAD

ORIG_HEAD

FETCH_HEAD

MERGE_HEAD

HEAD

Always refers to the most recent commit on the current branch.

When you change branches, HEAD is updated to refer to the new branch’s latest commit.

ORIG_HEAD

Certain operations, such as merge and reset, record the previous version of `HEAD` in `ORIG_HEAD` just prior to adjusting it to a new value.

You can use `ORIG_HEAD` to recover or revert to the previous state or to make a comparison.

FETCH_HEAD

When remote repositories are used, git fetch records the heads of all branches fetched in the file **.git/FETCH_HEAD** .

FETCH_HEAD is a shorthand for the head of the last branch fetched and is only valid immediately after a fetch operation.

MERGE_HEAD

When a merge is in progress, the tip of the other branch is temporarily recorded in the symref MERGE_HEAD .

MERGE_HEAD is the commit that is being merged into HEAD .

COMMITS

What is a commit

A commit is used to record changes to a repository.

When a commit occurs, Git records a “snapshot” of the index and places that snapshot in the object store.

What is a commit

A commit is the only method o introducing changes to a repository.

Commits are often introduced explicitly by a developer but Git can introduce commits (as in a merge).

Format

```
$ git commit -m “Message”

$ git commit -am “Message”

$ git commit -m “Message” <file>

$ git commit --amend
```

Atomic changesets

Every Git commit represents a single, atomic changeset with respect to the previous state.

Every changes apply in a commit or none (atomicity).

You can be assured that Git has not left your repository in some transitory state between one commit snapshot and the next.

Identifying Commits

Every Git commit is a hex-digit SHA1 ID.

Each commit ID is globally unique—not just for one repository, but for any and all repositories

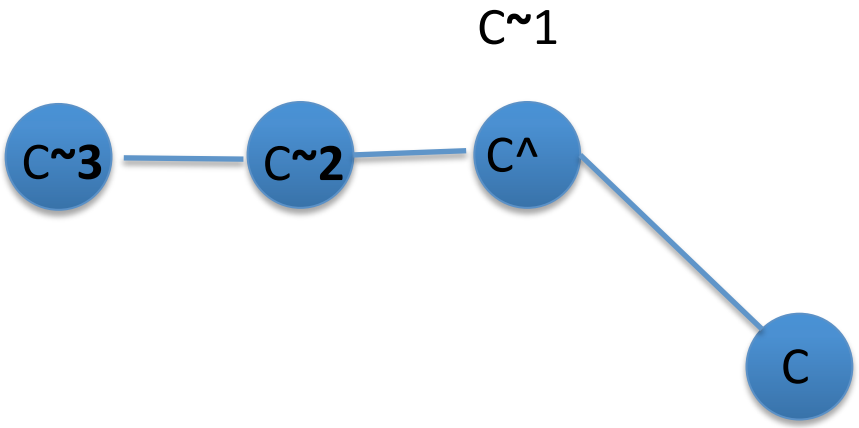
Relative Commit Names

Git provides mechanism for identifying a commit relative to another reference:

^: One commit before (master^, master^^, HEAD^,etc)

~: N commits before (master~2, HEAD~4)

Relative Commit Names



C is a commit

git show-branch --more=35

Commit history

You can see the history of commits:

$ git log (same as git log HEAD)

$ git log <commit-name>: The log starts at the named commit.

E.g: $ git log master

http://www.kernel.org/pub/software/scm/git/docs/v1.5.0.2/git-log.html

Specify a commit range (since..until)

$ git log master~12..master~10

Show only commits for a path

$ git log -- <path>

E.g. git log -- README3.txt

Show commits with a regular expression

$ git log --grep=‘reg-exp’

--no-merges: Removes merge commits

Show changes during a time

$ git log --since=“2.weeks.ago”

$ git log --before={3.weeks.ago} --after={2010-04-18}

Specify a format

--pretty=short: Short format

--abbrev-commit: SHA id abbreviation

-p: Prints the changes introduced by the commit

--stat: Enumerates the files changed in a commit

Show the graph

--decorate --graph --oneline

```
*   17e4c5f (HEAD, develop) Merge branch 'hotfix-1.0.1' into develop
|\
| * f43bb33 (hotfix-1.0.1) Hotfix2
| * 50a102d Hotifx1
| * d2fe7d6 Version 1.0.1
| *   d534111 (tag: 1.0) Merge branch 'release-1.0'
| |\
* | \   77e7eac Merge branch 'release-1.0' into develop
|\ \ \
| | |/
| |/|
| * | 9261fad (release-1.0) Release10Fix2
| * | 2562cb3 Release10Fix1
| * | c51b802 Version 1.0
* | | fbdd638 Work6
* | | 7353285 Work5
|/ /
* | 538bb9a Work4
* |   cdef254 Merge branch 'feature1' into develop
```

FIXING COMMITS

Fixing commits Flow

1. Discard changes in a file in the index to the copy in working directory

```
$ git checkout -- <files>
```

2. Reverting a file some commits before

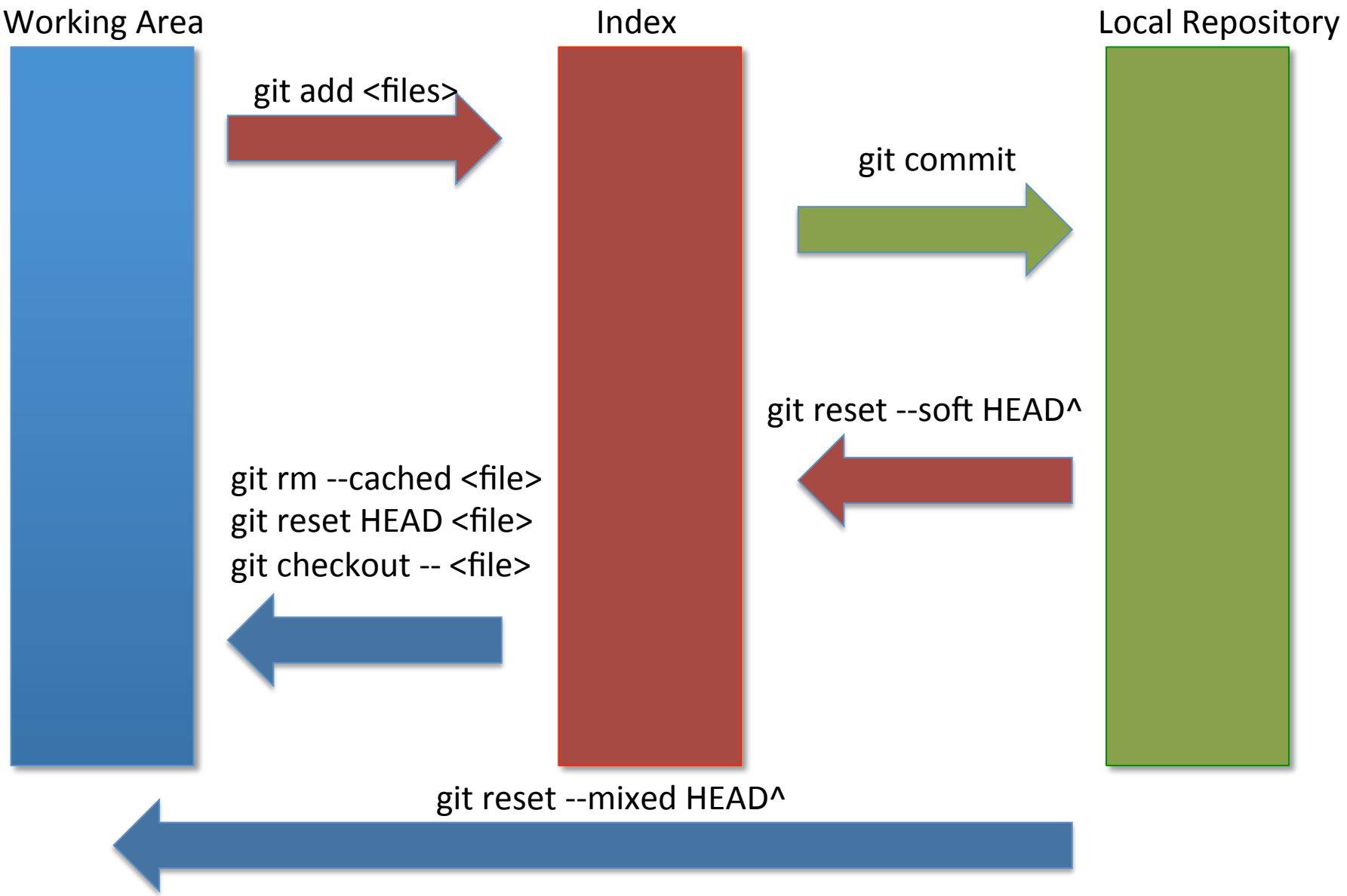
```
$ git checkout <commit-id> file
```

Basic Flow: Reset

You can reset some status of your repository:

```
$ git reset

$ git reset HEAD <file>

$ git reset --soft <commit-id>

$ git reset --hard <commit-id>

$ git reset --mixed <commit-id>
```

Basic Flow: Reset



Preguntas

Israel Alcázar
israelalcazar@gmail.com
@ialcazar