GIT version control

-- everything-is-local

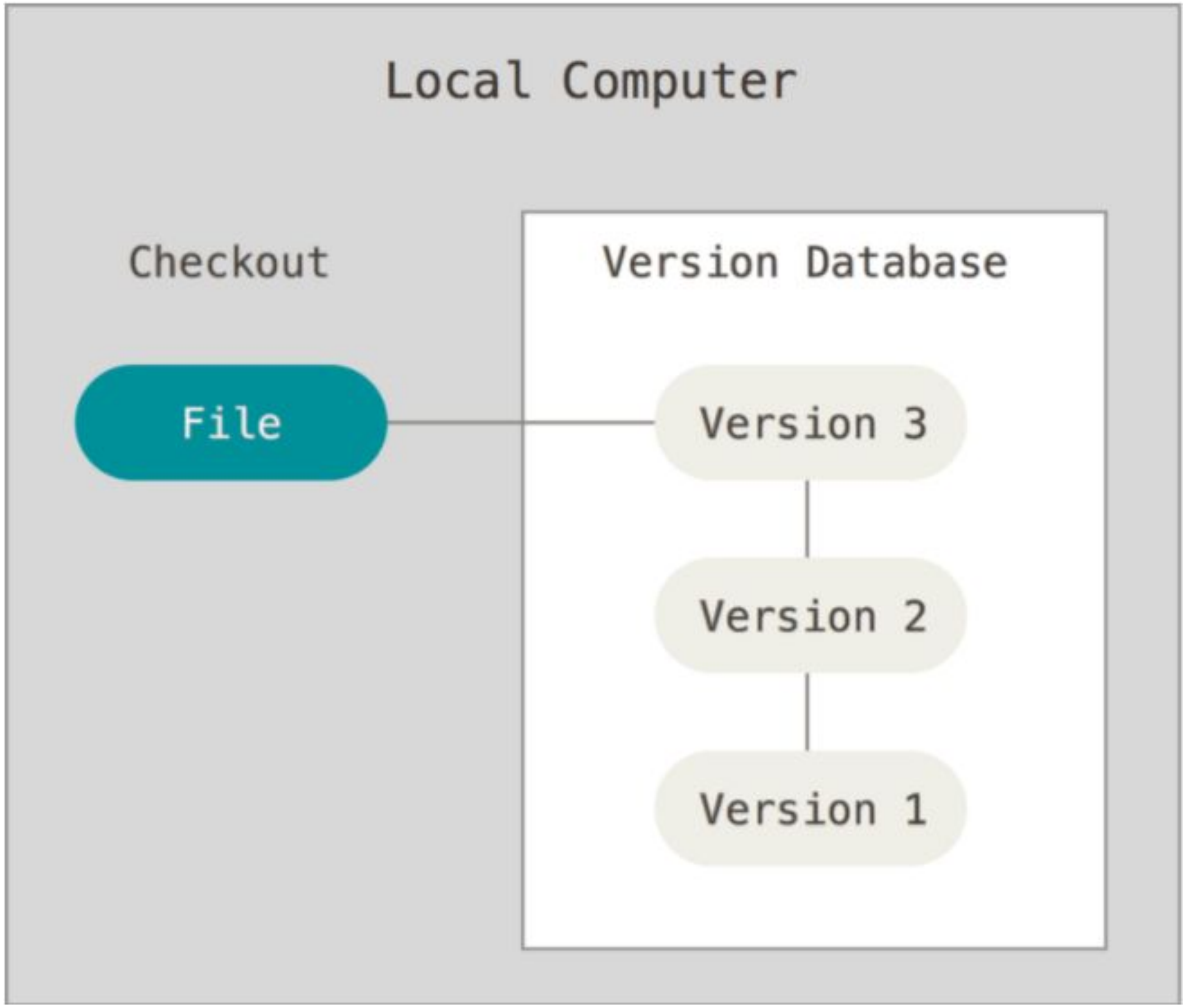
-- distributed-is-the-new-centralized
-- distributed-even-if-your-workflow-isn’t
--local-branching-on-the-cheap

- Sparsh Priyadarshi
GIT fan

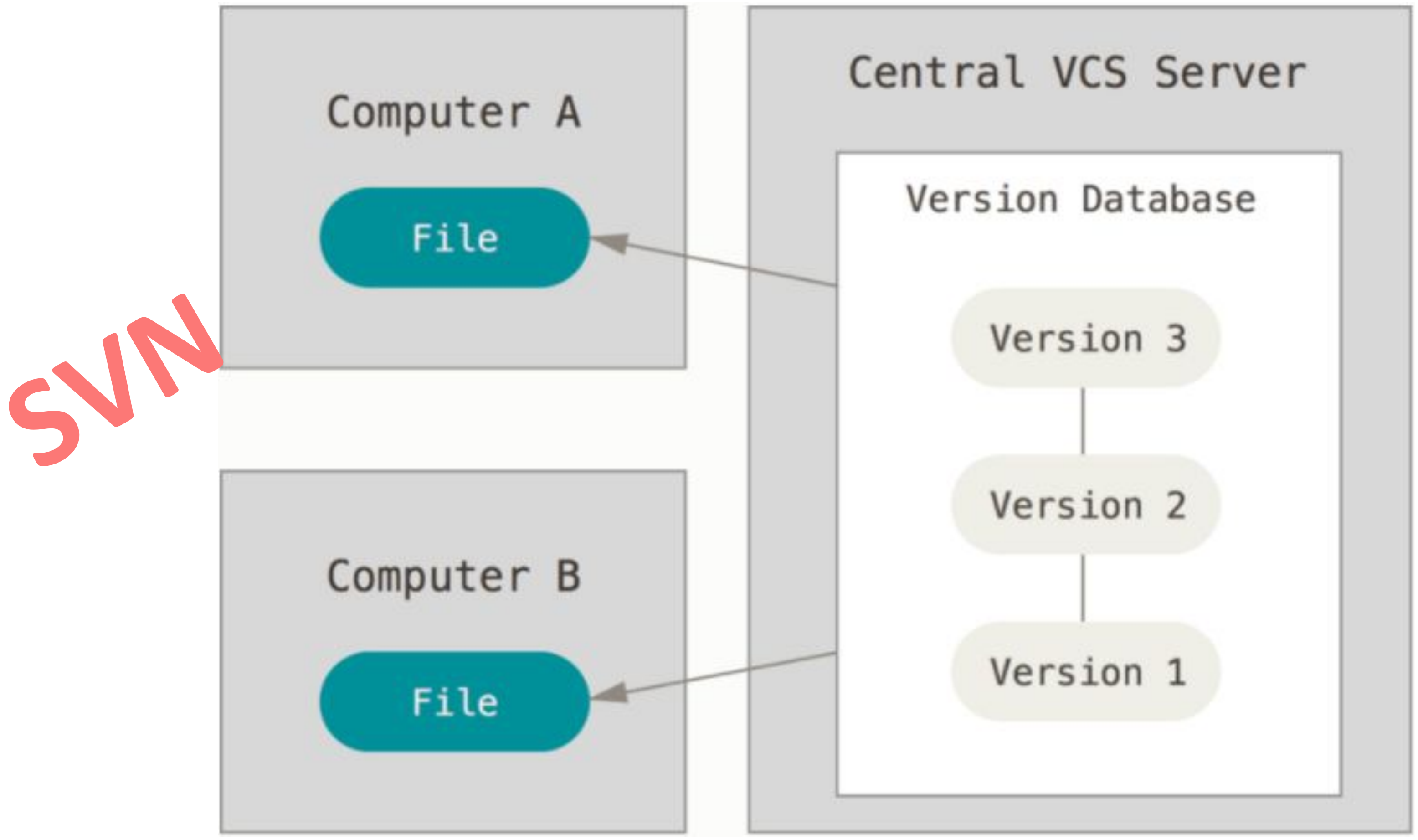
Bit about Version Control… what, why ?

Collaborate on code files, make changes, revert to previous version, add , remove modifications and most importantly Track all of this history.
And obviously do not overwrite on some one else' work 😊

Bit about version control…
Local Version Control

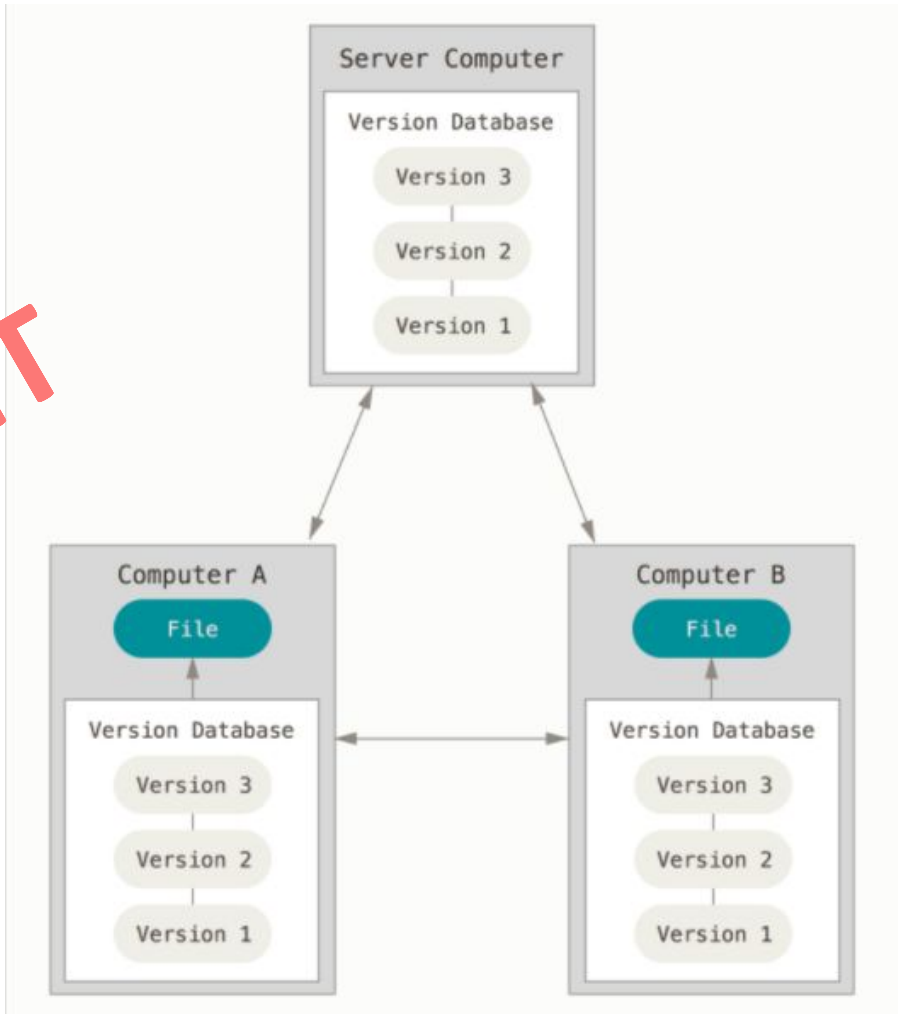


Bit about version control…
Centralized *Version Control*



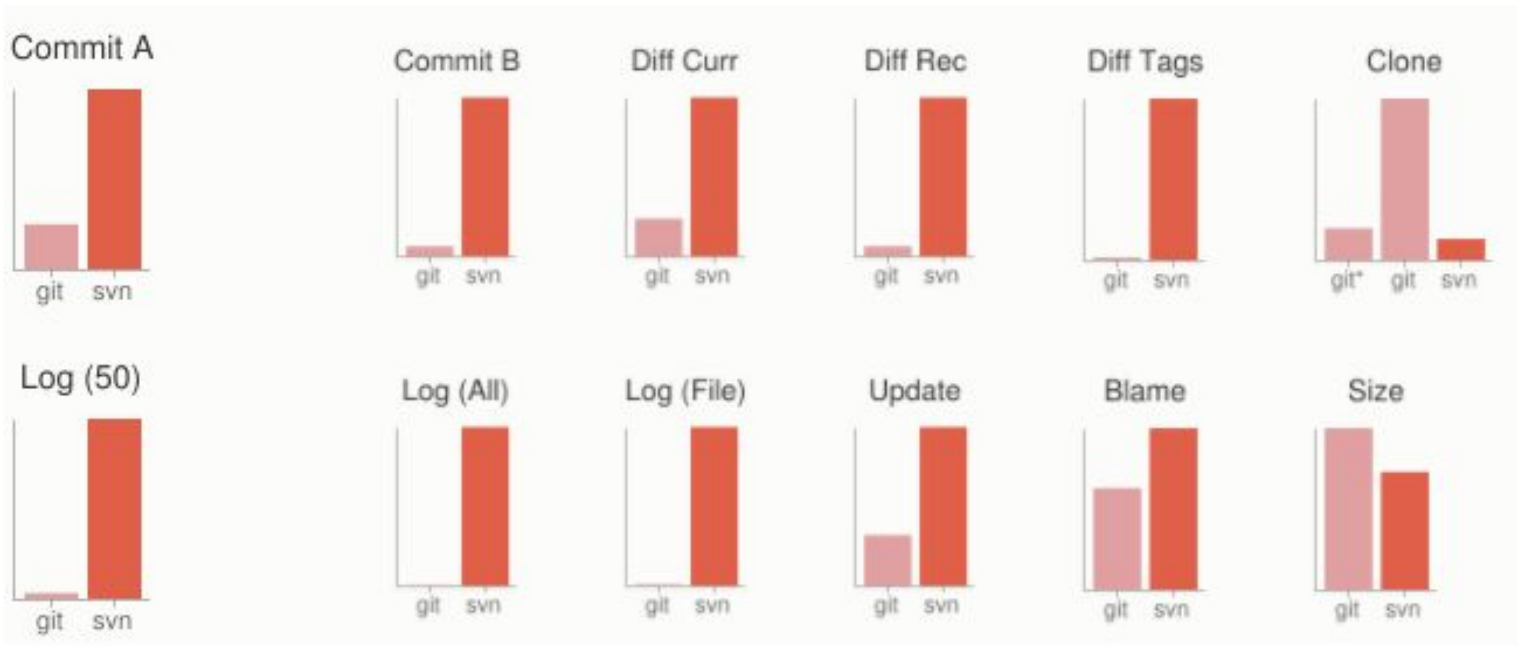
Bit about version control…
Distributed *Version Control*

GIT



GIT design goals and Advantages

Nearly all operations **local**
Was intended for linux codebase, hence **speed and performance** design goal from the start.

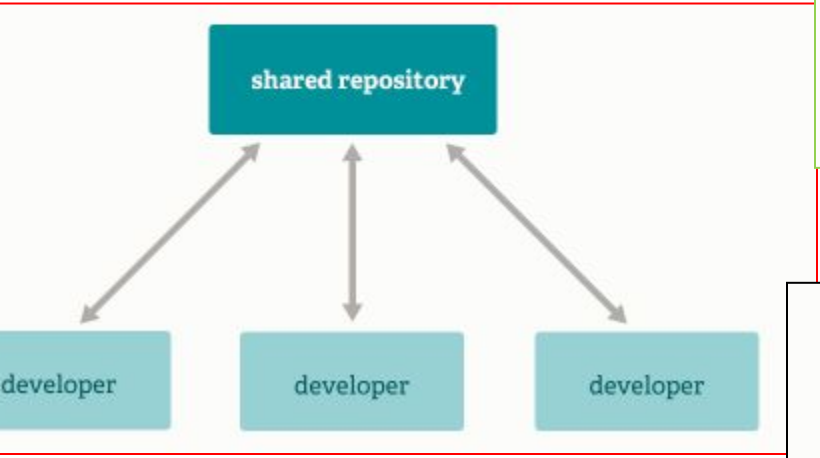


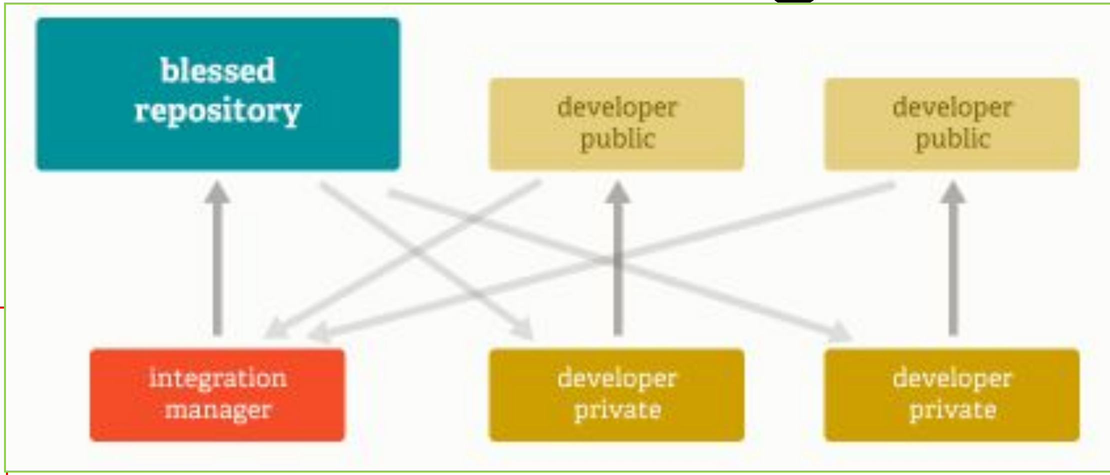
GIT design goals and Advantages

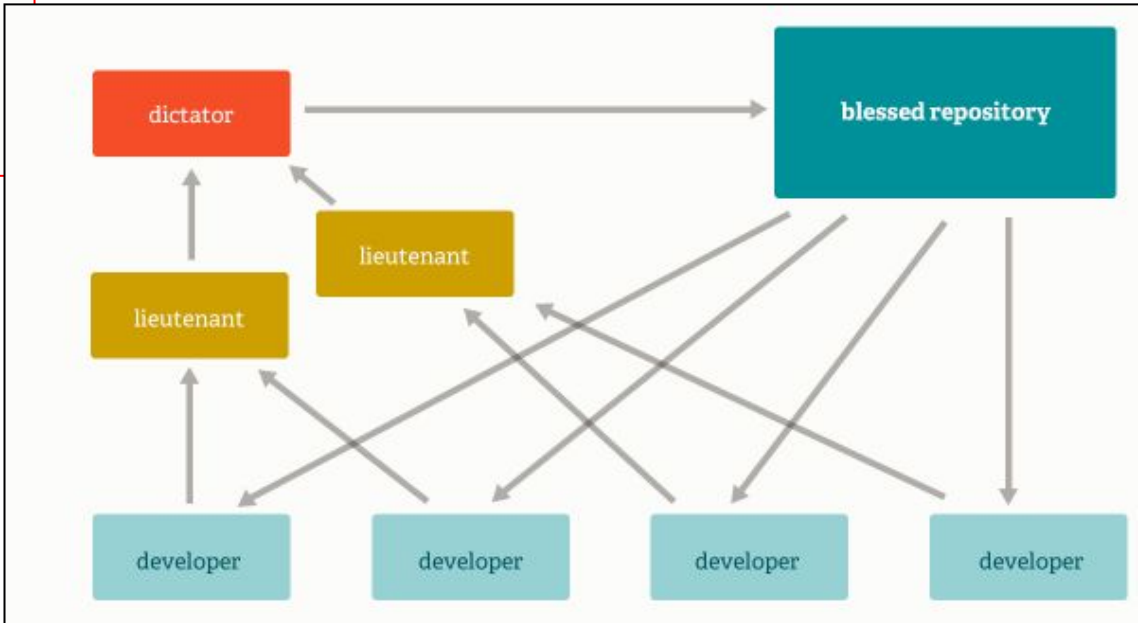
Distributed.

- Multiple backups
- Many workflows,
  SVN style, Integration Manager, Dictator – Lieutenants workflow

GIT design goals and Advantages



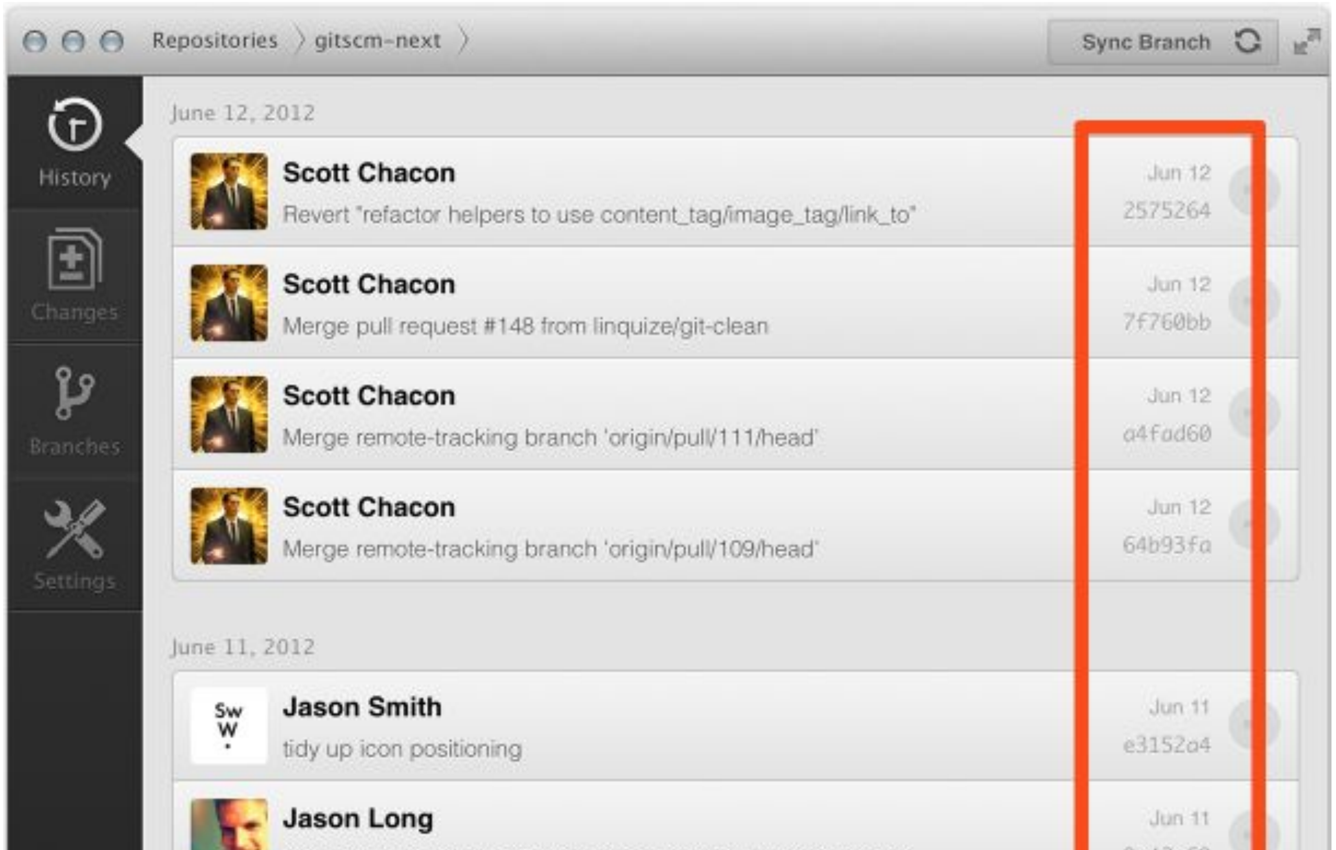




GIT design goals and Advantages

Data Assurance.

cryptographic integrity



June 12, 2012

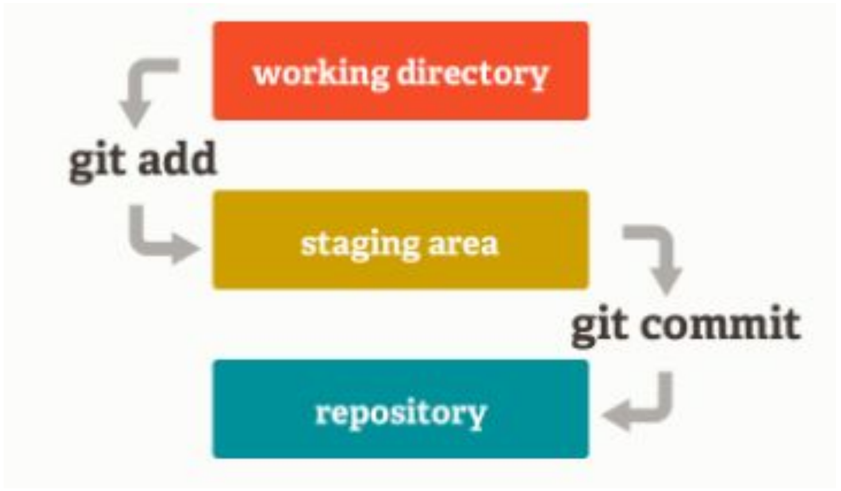
Scott Chacon	Revert "refactor helpers to use content_tag/image_tag/link_to"	Jun 12 2575264
Scott Chacon	Merge pull request #148 from linquize/git-clean	Jun 12 7f760bb
Scott Chacon	Merge remote-tracking branch 'origin/pull/111/head'	Jun 12 a4fad60
Scott Chacon	Merge remote-tracking branch 'origin/pull/109/head'	Jun 12 64b93fa

June 11, 2012

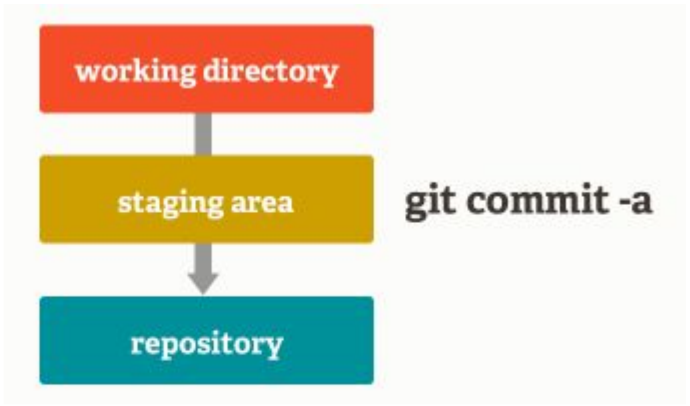
Jason Smith	tidy up icon positioning	Jun 11 e3152a4
Jason Long		Jun 11

GIT design goals and Advantages

Staging Area



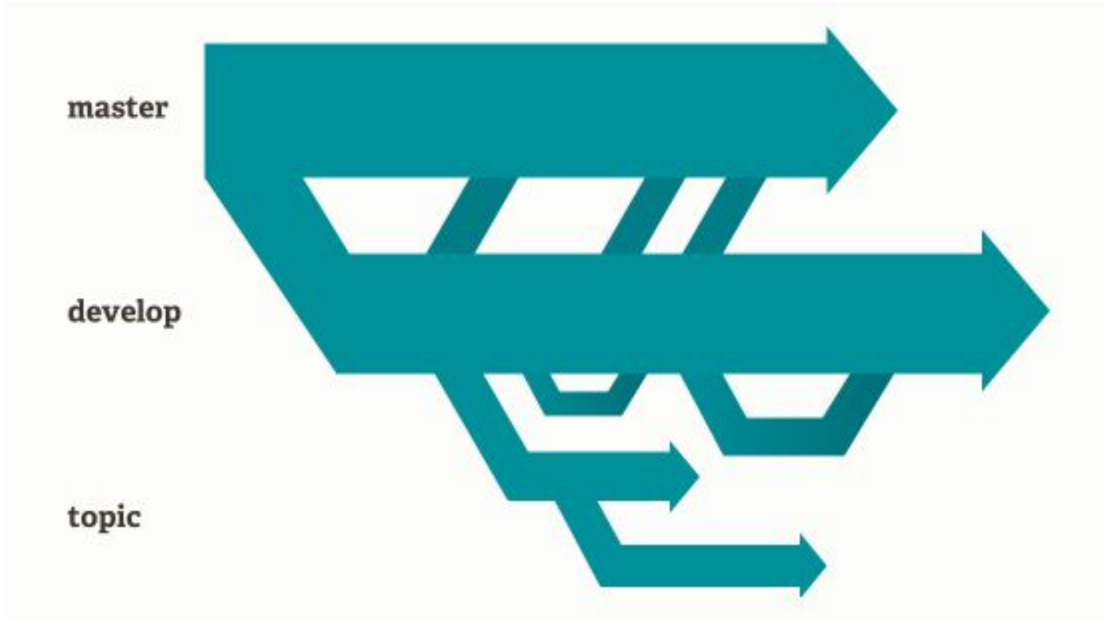
If don't like then bypass…



GIT design goals and Advantages

Branching and Merging

- Frictionless Context Switching
- Role-Based Codelines.
- Feature Based Workflow.
- Disposable Experimentation.



Think like GIT design goals and Advantages

Free and Open Source, yay !

Think Like GIT (and not SVN)

Snapshots, Not Differences

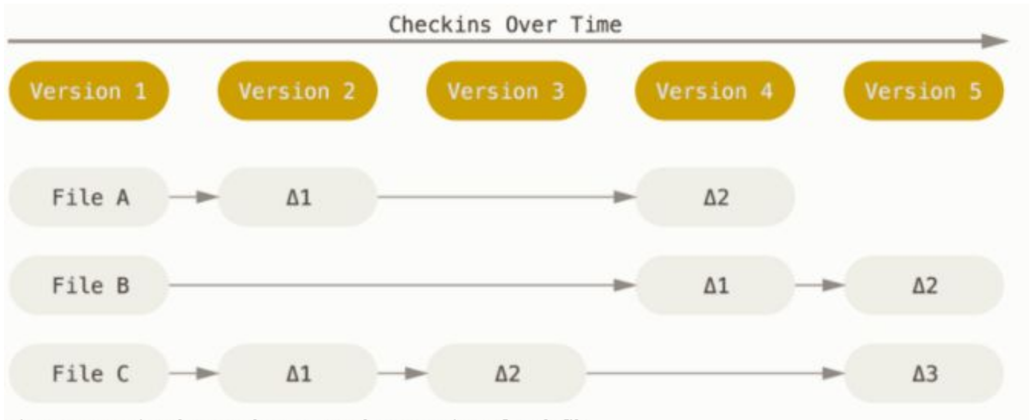


Figure 4. Storing data as changes to a base version of each file.

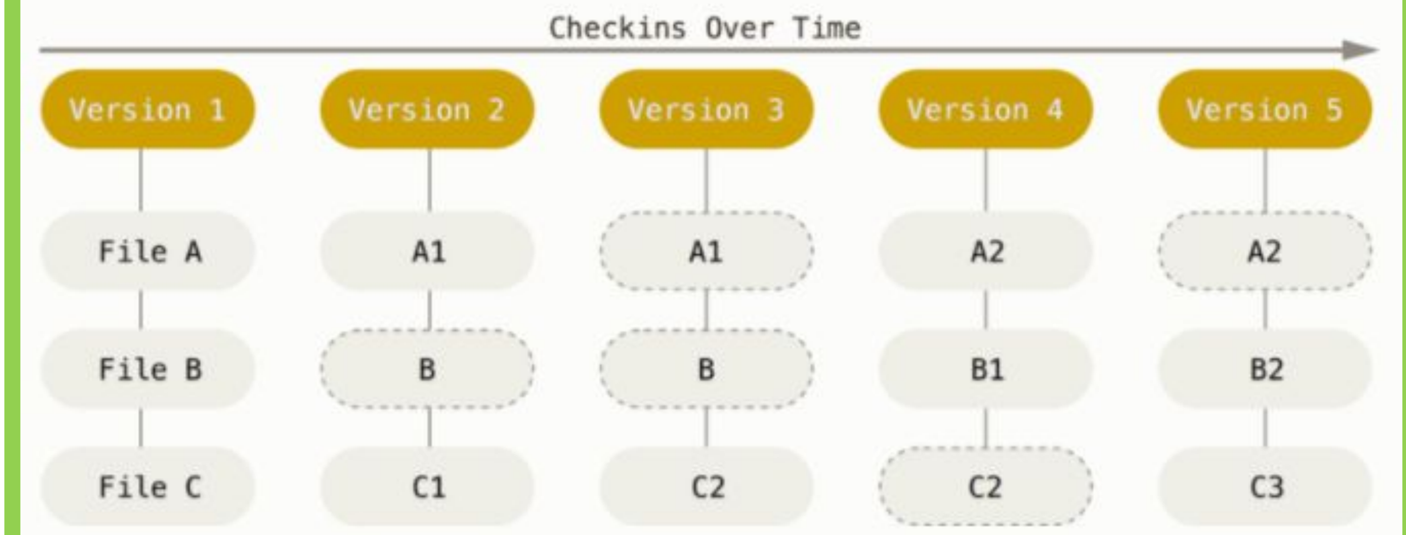


Figure 5. Storing data as snapshots of the project over time.

Think Like GIT (and not SVN)

***Nearly* Every Operation Is Local**

… Entire history since you took (cloned) repository is present locally,
make commits, merge branches etc.
no network needed.

Integrity : everything is checksummed with SHA1 impossible to change anything without GIT detecting it.

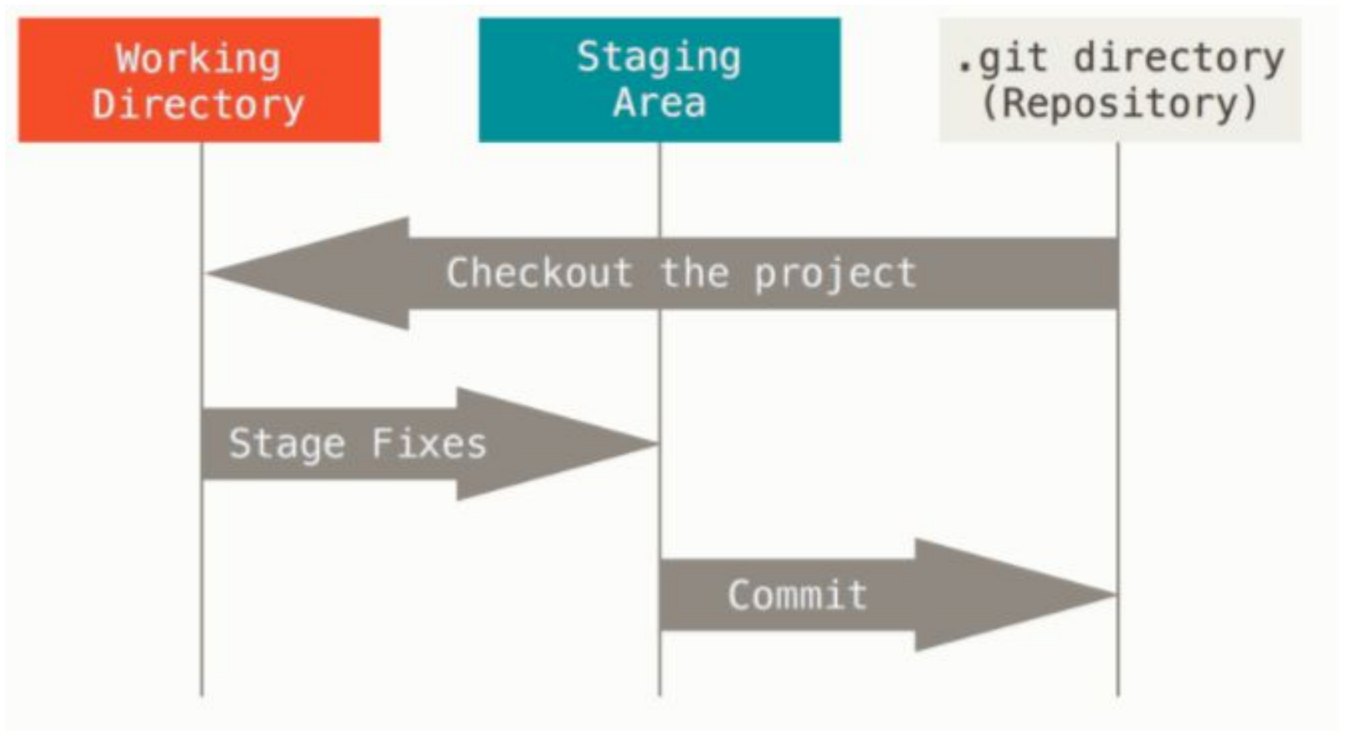
GIT in nutshell…

"I know… enough history lessons "😊

modified staged committed

^ Stages that a file can exist in git ^

GIT in nutshell…



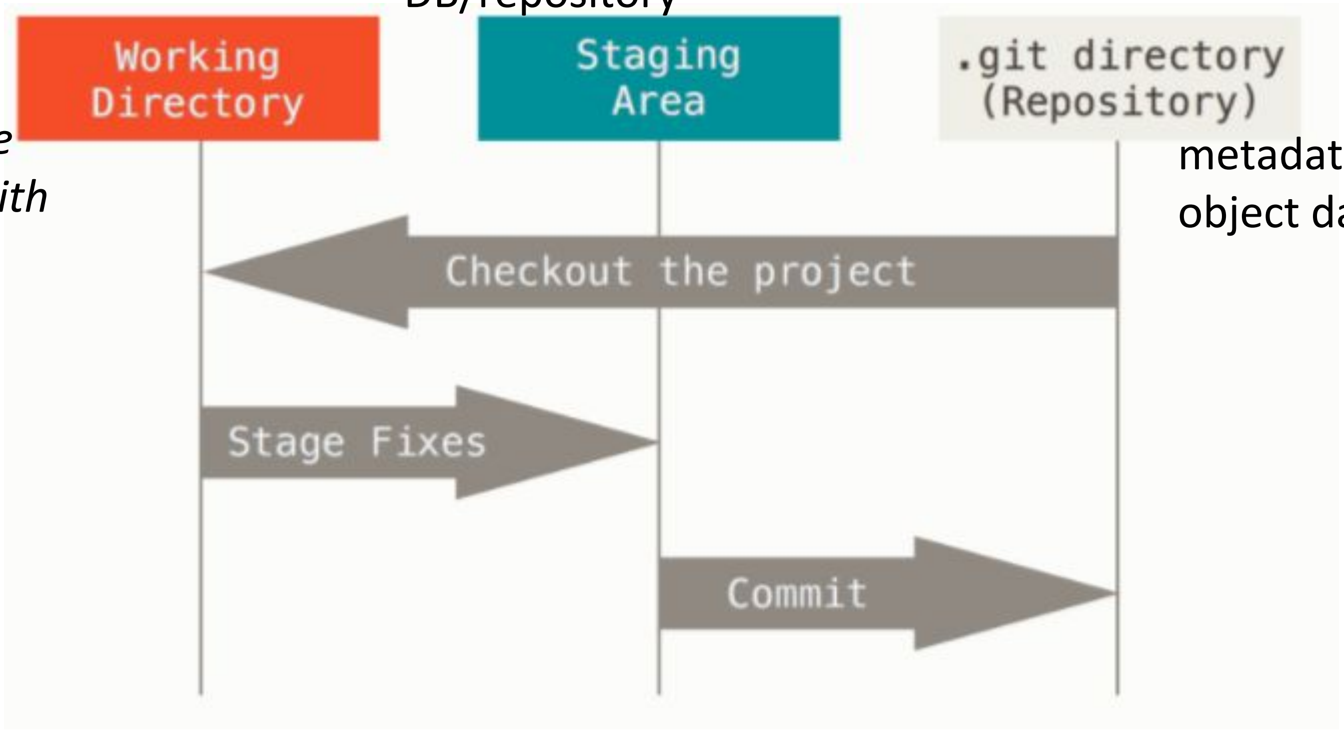
modified staged committed

^ Stages that a file can exist in git ^

file that stores information about what will go into your next commit -> snapshot ->local DB/repository

single checkout of one version

what we see and work with



metadata & object database

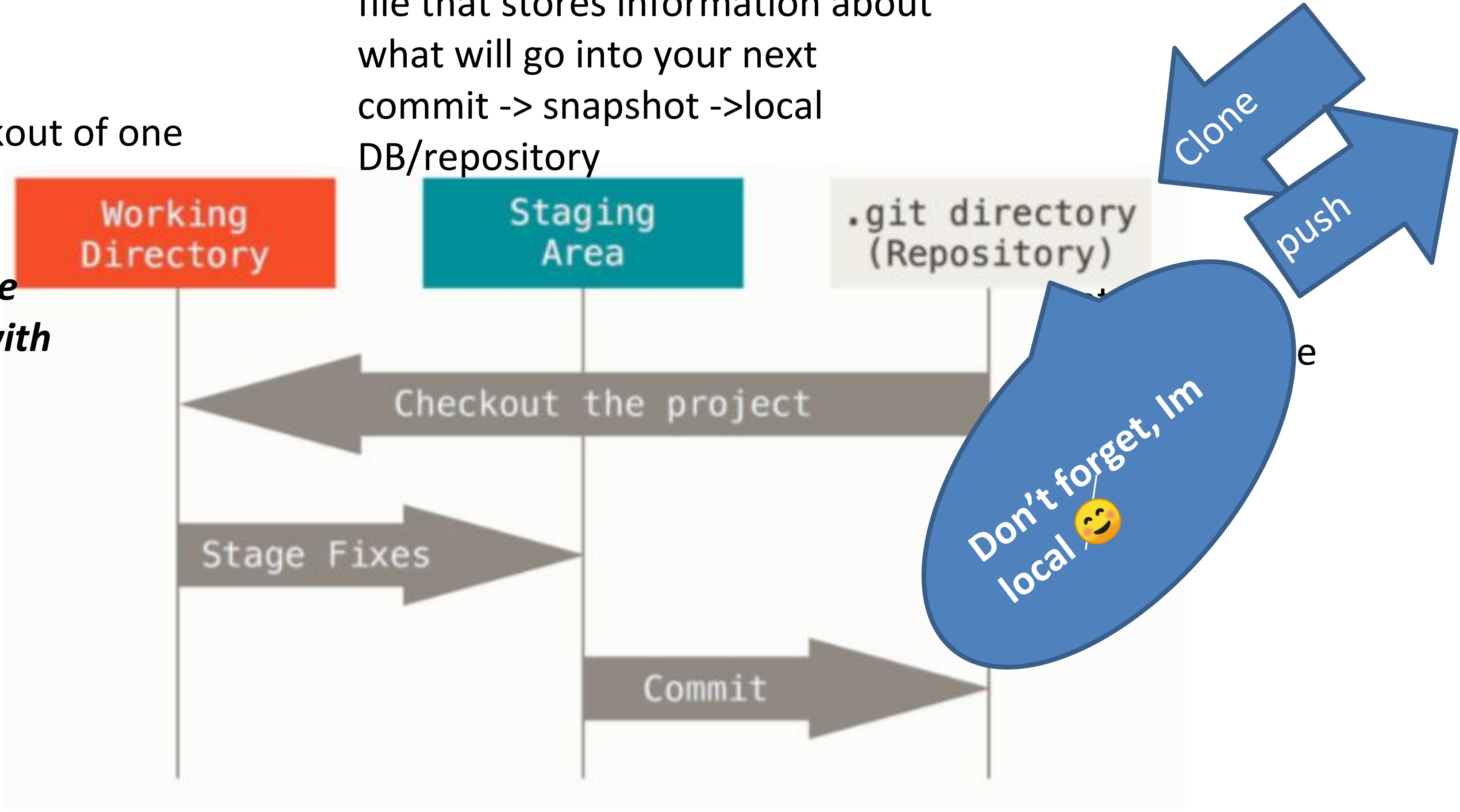
modified staged committed

^ Stages that a file can exist in git ^

file that stores information about what will go into your next commit -> snapshot ->local DB/repository

single checkout of one version

what we see and work with



modified staged committed

^ Stages that a file can exist in git ^

Getting started

-- install
-- git-bash / git-cmd / git-GUI

GUIs for GIT
(tortoiseGIT, githubDesktop, SourceTree etc…)

---

git config

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
$ git config --list
```

git help <git-command>

Get a Git repository

In an existing directory

```
git init
```

Cloning an existing repo from server.

```
git clone [url]
```

Recording changes to a Git repository

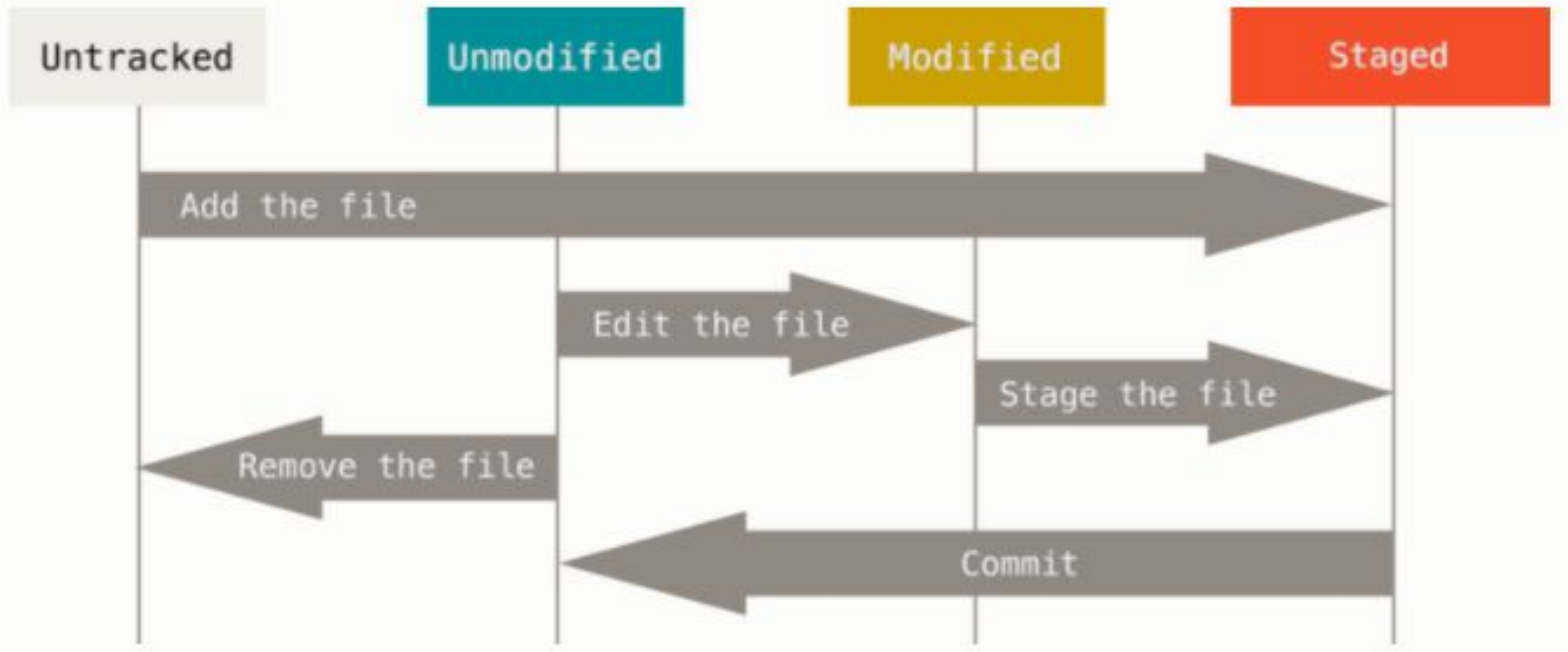


Figure 8. The lifecycle of the status of your files.

Recording changes to a Git repository

git status
git add
.gitignore file
git diff
git diff --staged
git commit
git rm
git rm –cached
git mv

Viewing history in Git

```
git log
```

Undoing things

To amend previous commit

```
git commit --amend
```

To unstage

```
git reset HEAD <file>
```

To discard a modified file

```
git checkout -- <file>
```

Working with Remotes

```
git remote [-v]
git remote add <shortname> <url>
git fetch <remotename>
git pull // = fetch+merge from remote
git push [remote-name] [branch-name]
git remote show <remotename>
git remote rename <old> <new>
git remote remove <remotename>
```

Branching

“Unlike many other VCSs, Git encourages workflows that branch and merge often, even multiple times in a day. Understanding and mastering this feature gives you a powerful and unique tool and can entirely change the way that you develop.”

-- wise words in the GIT documentation

Branching

```
git branch [-v]
git branch <branchname>
Git branch –d <branchname>
HEAD pointer
git checkout <branchname>
```

Merging

```
git merge <branch-to-merge-in>
```

Nearing end…

`(Workflow review)`

Appendix ?

..git rebase

..remote tracking advanced concepts, remote branch management, configuring multiple remotes for fetch/push

..git on server -> configuring gitlab, github etc. configuring read/write access, ssh keys …

..tags

..aliases

..commit stashing

Tags

```
git tag
```

Aliases

Eg.

```
git config --global alias.ci commit
```

```
>>ci –m “commitmessage”
```