Git, GitHub, EGit

by- **Saladin Minhaaj**

git

Version Control System

- a system that records changes to a file or set of files over time so that you can recall specific versions later

Why VCS ?

- revert files back to a previous state
- revert the entire project back to a previous state
- review changes made over time
- see who last modified something that might be causing a problem
- who introduced an issue and when

Why VCS ?

- if you screw things up or lose files, you can generally recover easily
- In addition, you get all this for very little overhead
- and many more

VCS Types

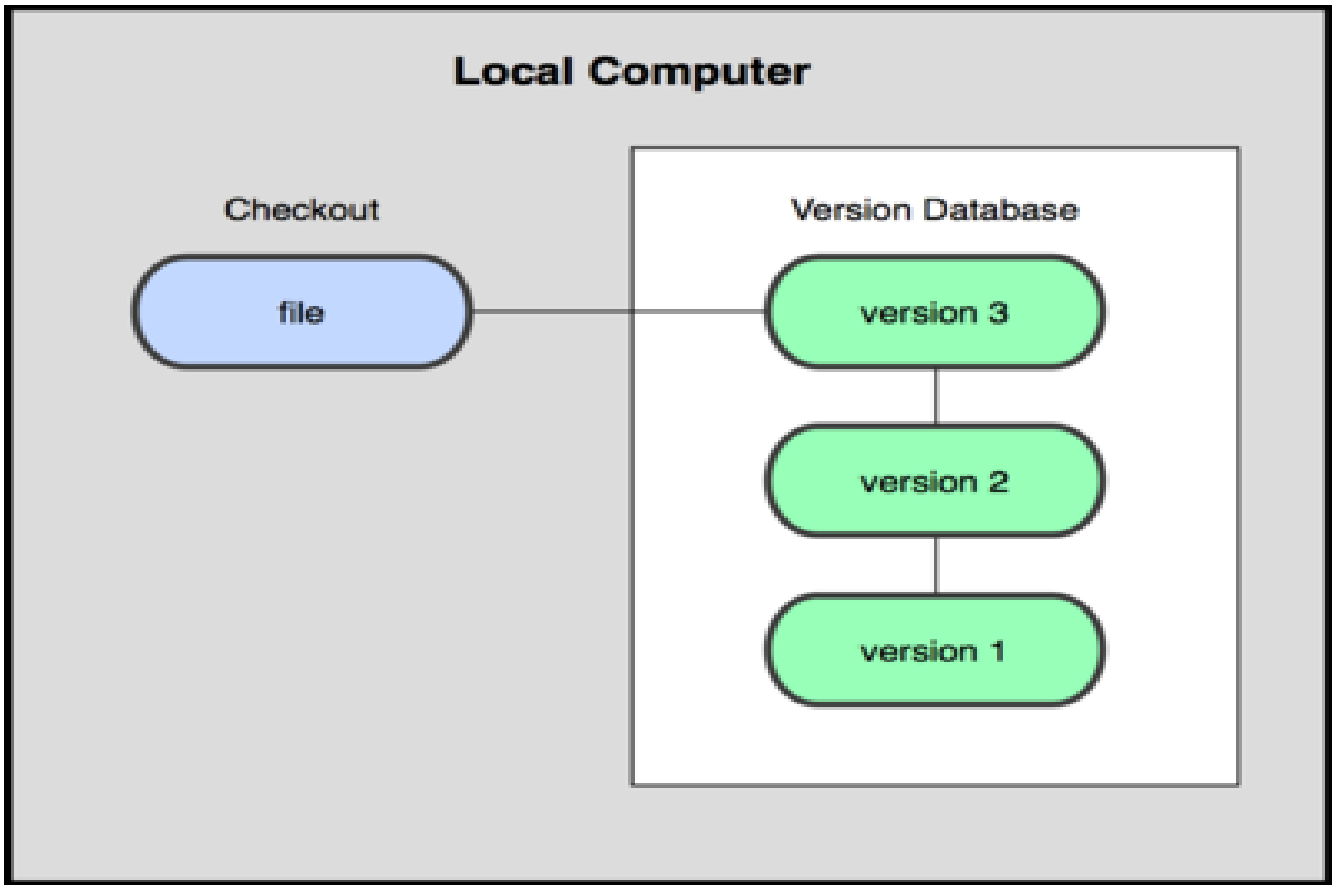
- Local VCS
- Centralized VCS
- Distributed VCS

Personal Version Control Method

- copy files into another directory
- +ve: so simple
- -ve: incredibly error-prone
- It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to

Local VCS

- Use a simple database that keeps all the changes to files under revision control



Local VCS

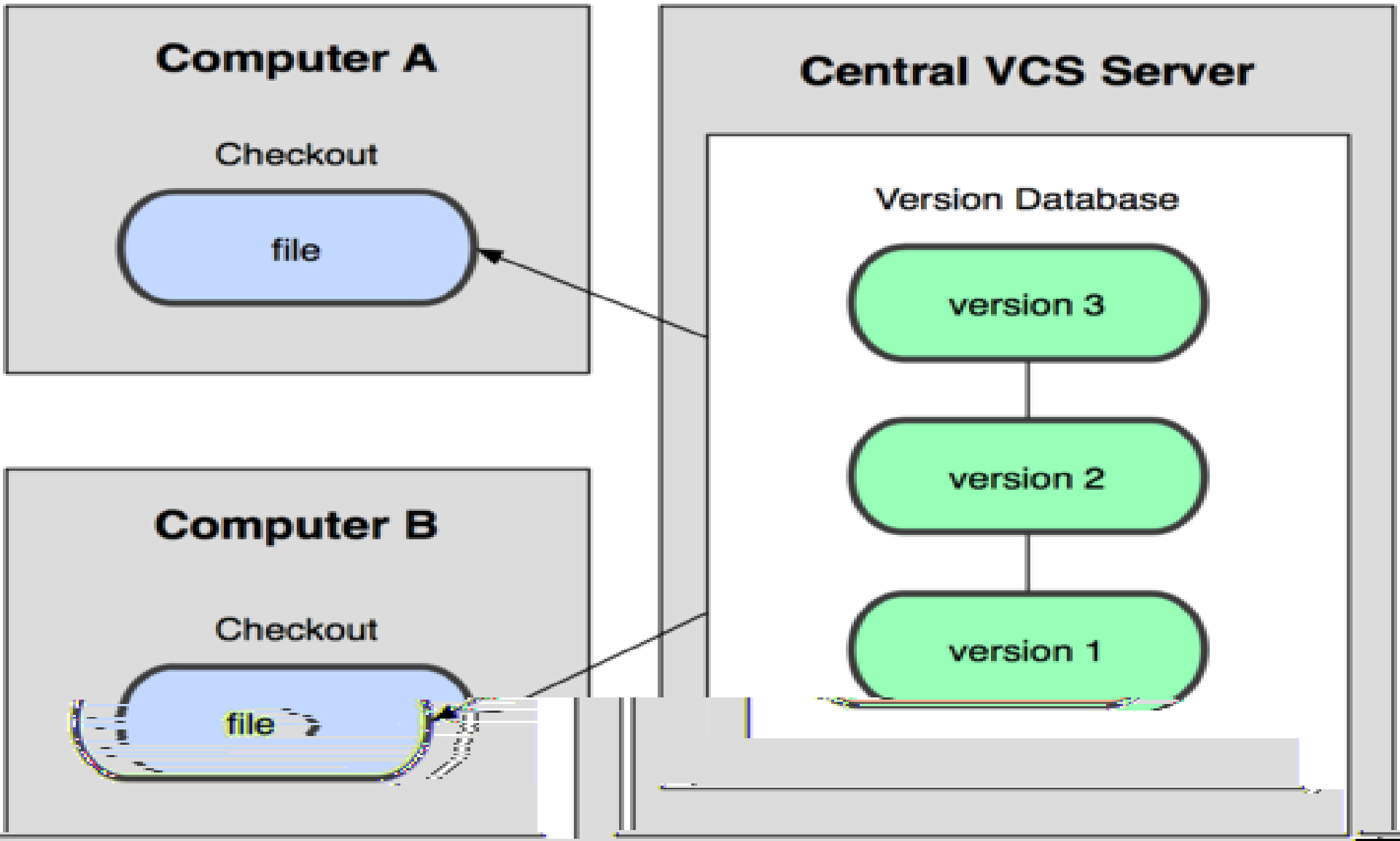
- Issues:

1. Confined within a single computer
2. Hard disk becomes corrupted

Centralized VCS

- collaboration with developers on other systems
- e.g. CVS, Subversion, and Perforce
- Centralized VCSs have a single server that contains all the versioned files, and a number of clients that check out files from that central place

Centralized VCS



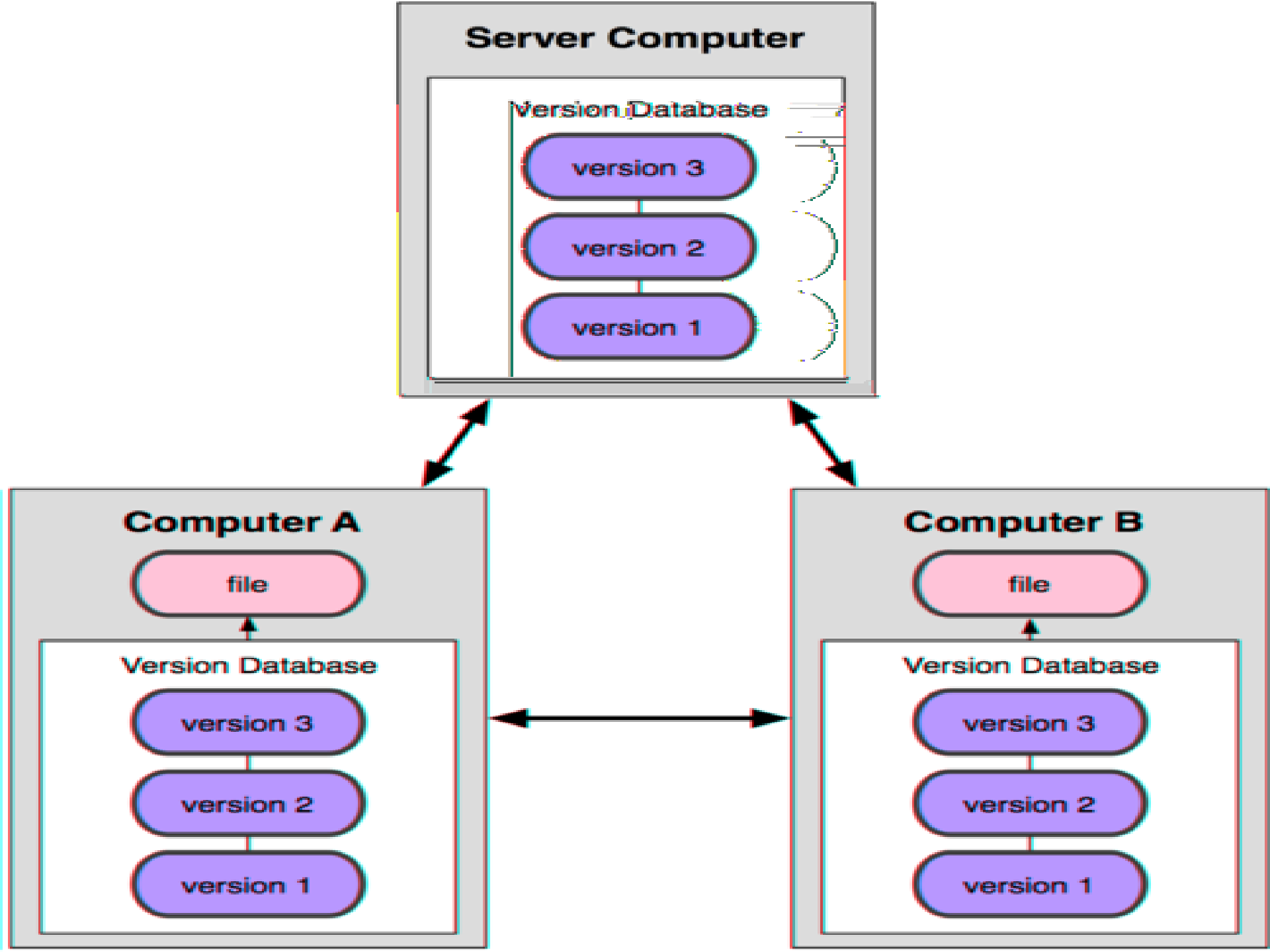
Centralized VCS

- Issues:

1. Server goes down
2. Hard disk of the central database becomes corrupted

Distributed VCS

- Clients don’t just checkout the files: they checkout the entire repository
- If any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it
- Every checkout is really a full backup of all the data

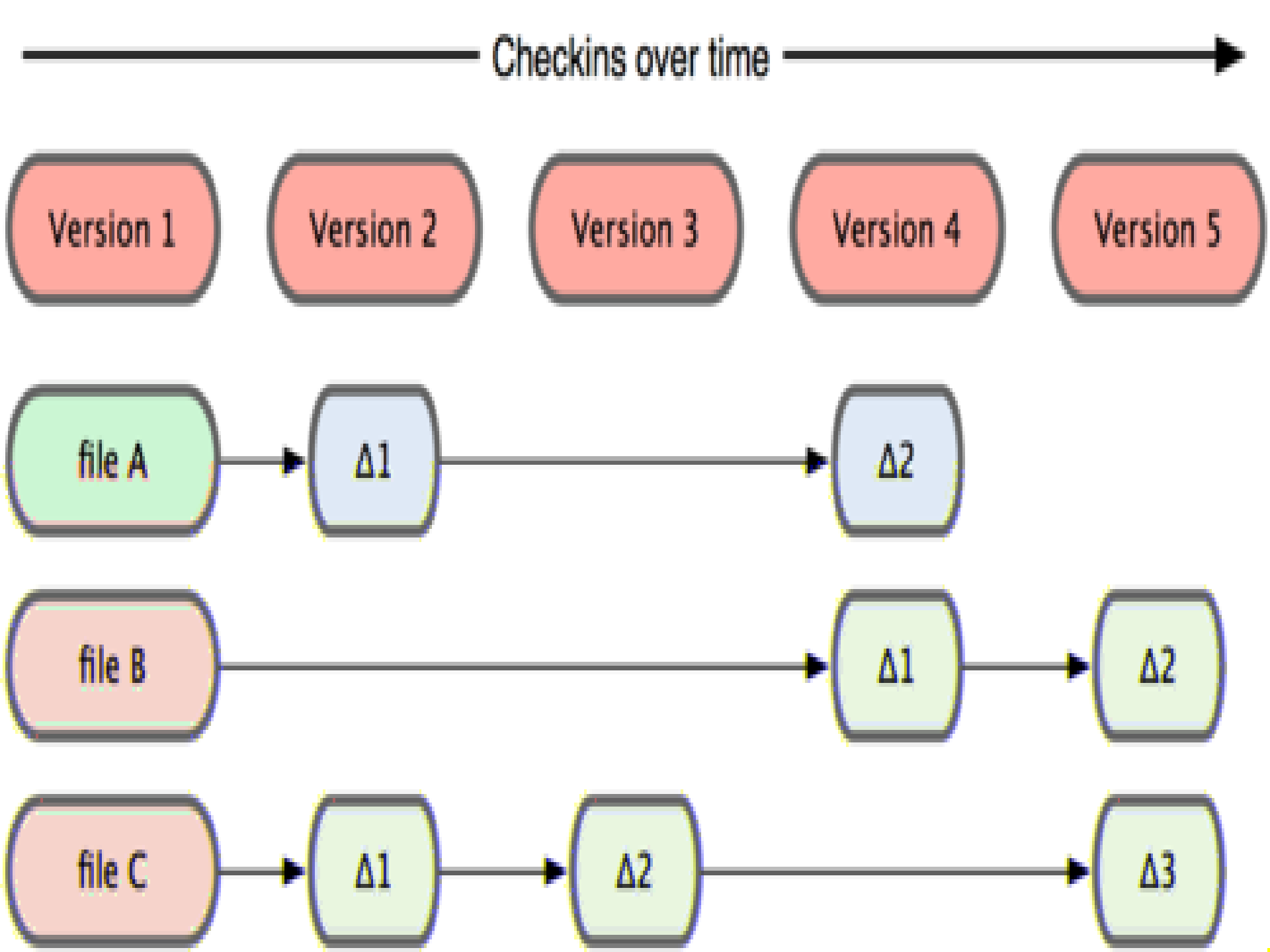

Server Computer
Version Database
version 3
version 2
version 1
Computer A
file
Version Database
version 3
version 2
version 1
Computer B
file
Version Database
version 3
version 2
version 1

Uniqueness of Git

- Git mechanism
- Local operations
- Integrity
- Addition of data only

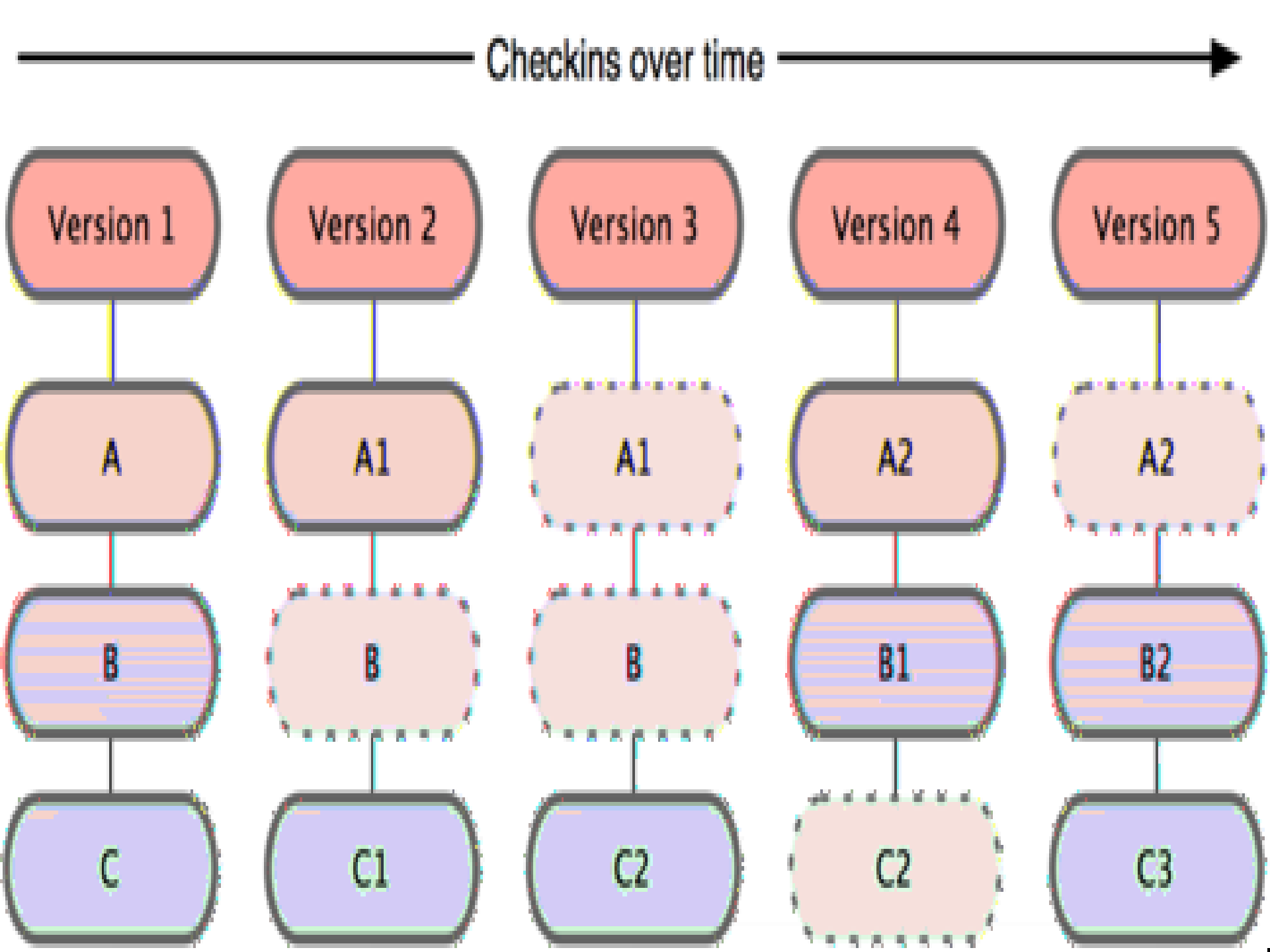
Common VCS Mechanism

- Conceptually, most other systems store information as a list of file-based changes
- These systems think of the information they keep as a set of files and the changes made to each file over time


Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
file A
Δ1
Δ2
file B
Δ1
Δ2
file C
Δ1
Δ2
Δ3

Git Mechanism

- Git doesn’t store its data this way
- Instead, Git thinks of its data more like a set of snapshots of a mini file-system
- Every time a client commits, or saves the state of his project in Git, it basically takes a picture of what all his files look like at that moment and stores a reference to that snapshot
- To be efficient, if files have not changed, Git doesn’t store the file again—just a link to the previous identical file it has already stored


Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
A
A1
A1
A2
A2
B
B
B
B1
B2
C
C1
C2
C2
C3

Local Operations

- Most operations in Git only need local files and resources to operate — generally no information is needed from another computer on your network

Integrity

- Everything in Git is check-summed before it is stored and is then referred to by that checksum
- It’s impossible to change the contents of any file or directory without Git knowing about it
- This functionality is built into Git at the lowest levels and is integral to its philosophy
- You can’t lose information in transit or get file corruption without Git being able to detect it

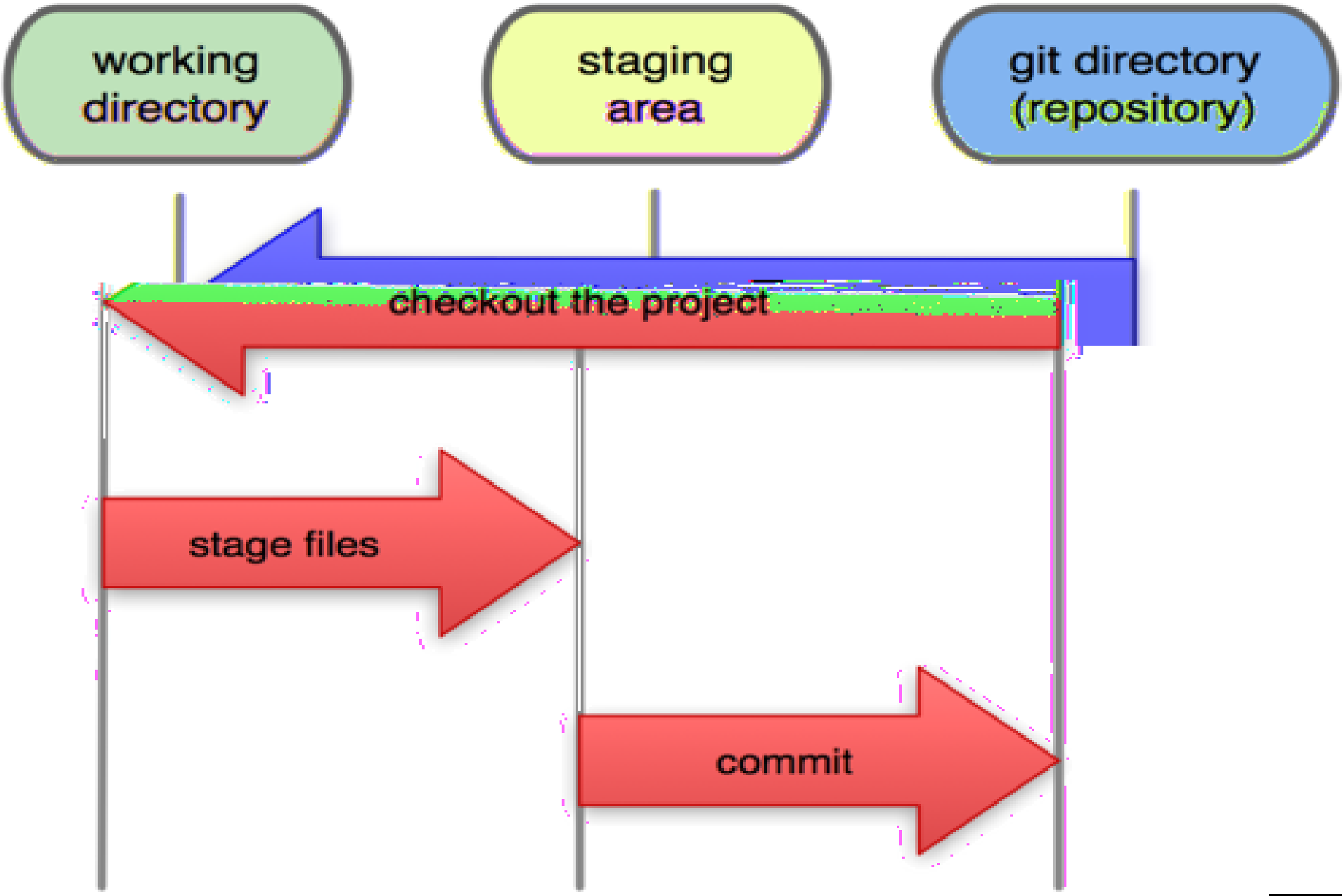
Integrity

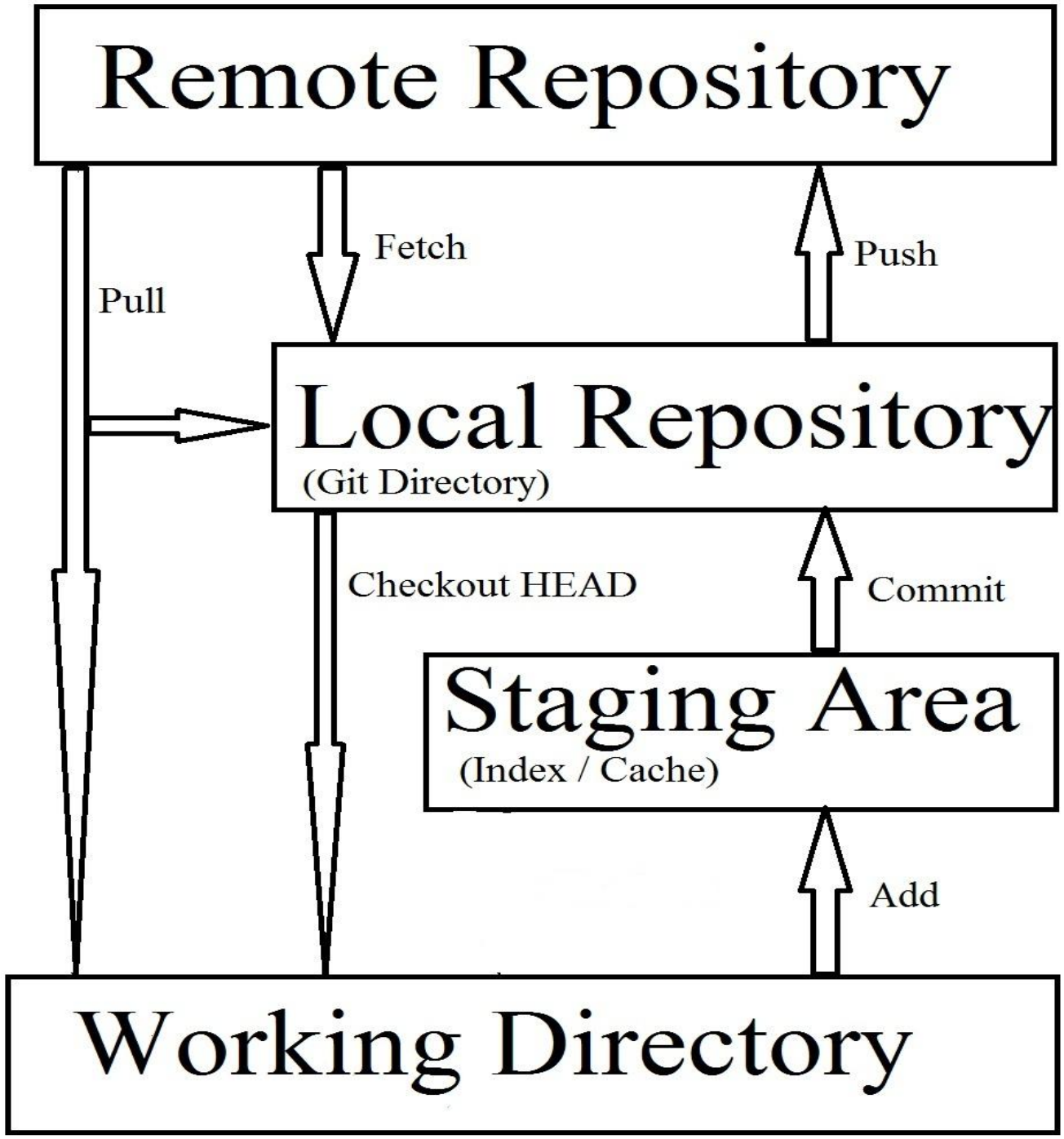
- The mechanism that Git uses for this check-summing is called a SHA-1 hash
- This is a 40-character string composed of hexadecimal characters and calculated based on the contents of a file or directory structure in Git
- A SHA-1 hash looks something like this:
  24b9da6552252987aa493b52f8696cd6d3b00373
- Git stores everything not by file name but in the Git database addressable by the hash value of its contents.

Addition of data only

- When you do actions in Git, nearly all of them only add data to the Git database
- It is very difficult to get the system to do anything that is not undoable or to make it erase data in any way
- As in any VCS, you can lose or mess up changes you haven't committed yet; but after you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository

Local Operations




Remote Repository
Fetch
Push
Pull
Local Repository
(Git Directory)
Checkout HEAD
Commit
Staging Area
(Index / Cache)
Add
Working Directory

Install on Windows

- Simply download the installer .exe file from the GitHub page, and run it:

  http://msysgit.github.com/

- Download from GitHub : my preferred way

Install on Mac

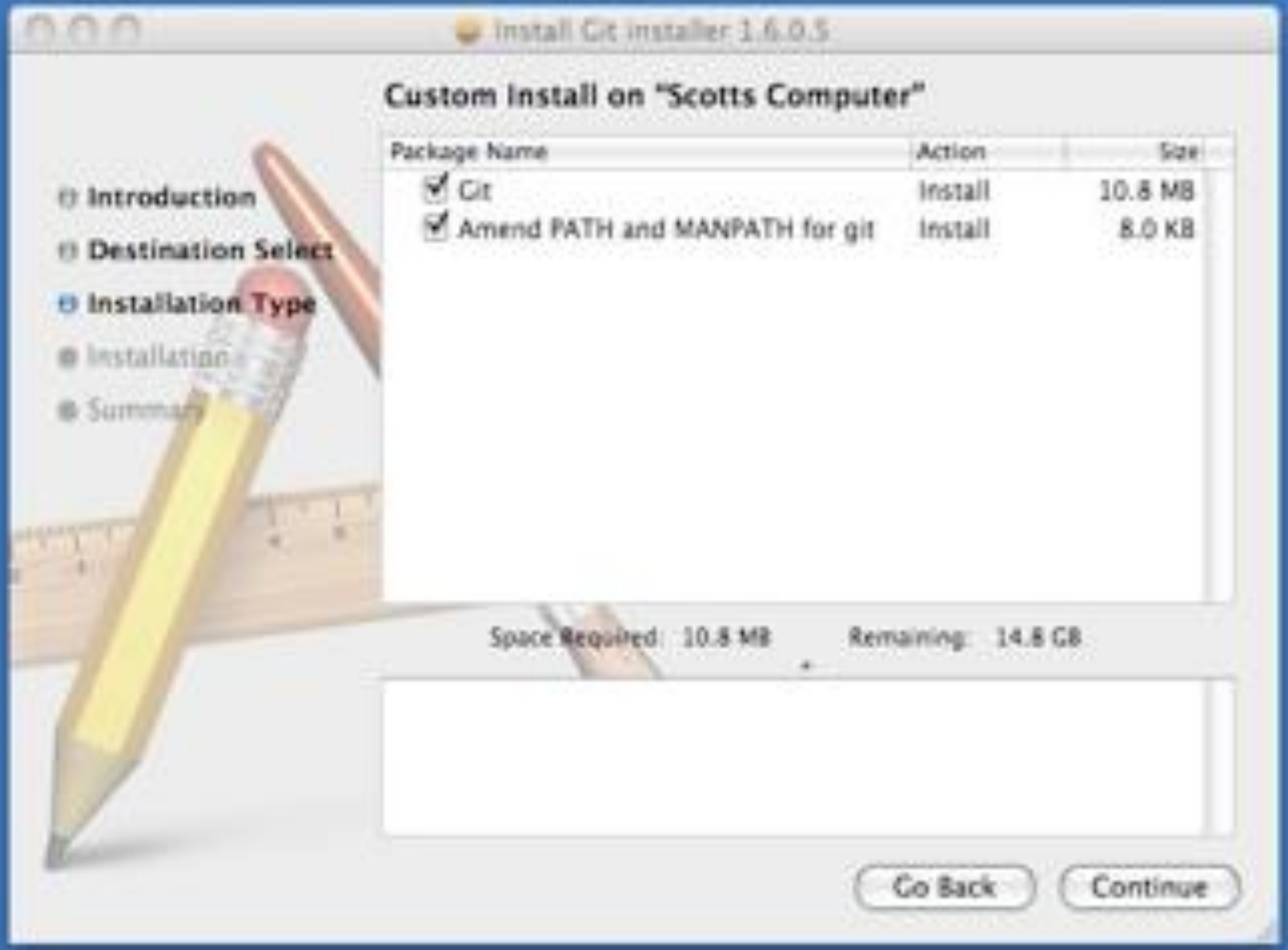
1. to use the graphical Git installer, which you can download from: http://code.google.com/p/git-osx-installer

2. via MacPorts (http://www.macports.org). If you have MacPorts installed, install Git via

$ sudo port install git-core +svn +doc +bash_completion +gitweb

You don't have to add all the extras, but you'll probably want to include +svn in case you ever have to use Git with Subversion repositories

3. Download from GitHub: my preferred way


Install Git installer 1.6.0.5
Custom Install on "Scotts Computer"
Introduction
Destination Select
Installation Type
Installation
Summary
Package Name
Action
Size
Git
Install
10.8 MB
Amend PATH and MANPATH for git
Install
8.0 KB
Space Required: 10.8 MB
Remaining: 14.8 GB
Go Back
Continue

Git Configuration

- Identity set up:

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

- Editor set up

```
git config --global core.editor notepad
```

Git Configuration

- Check all the settings:

```
git config --list
```

- Check specific setting:

```
git config user.name
```

Git Configuration

- Getting help:

git help <command>

git <command> --help

man git-<command>

e.g.

git help config

Git commands

- Initializing a local repository

  git init

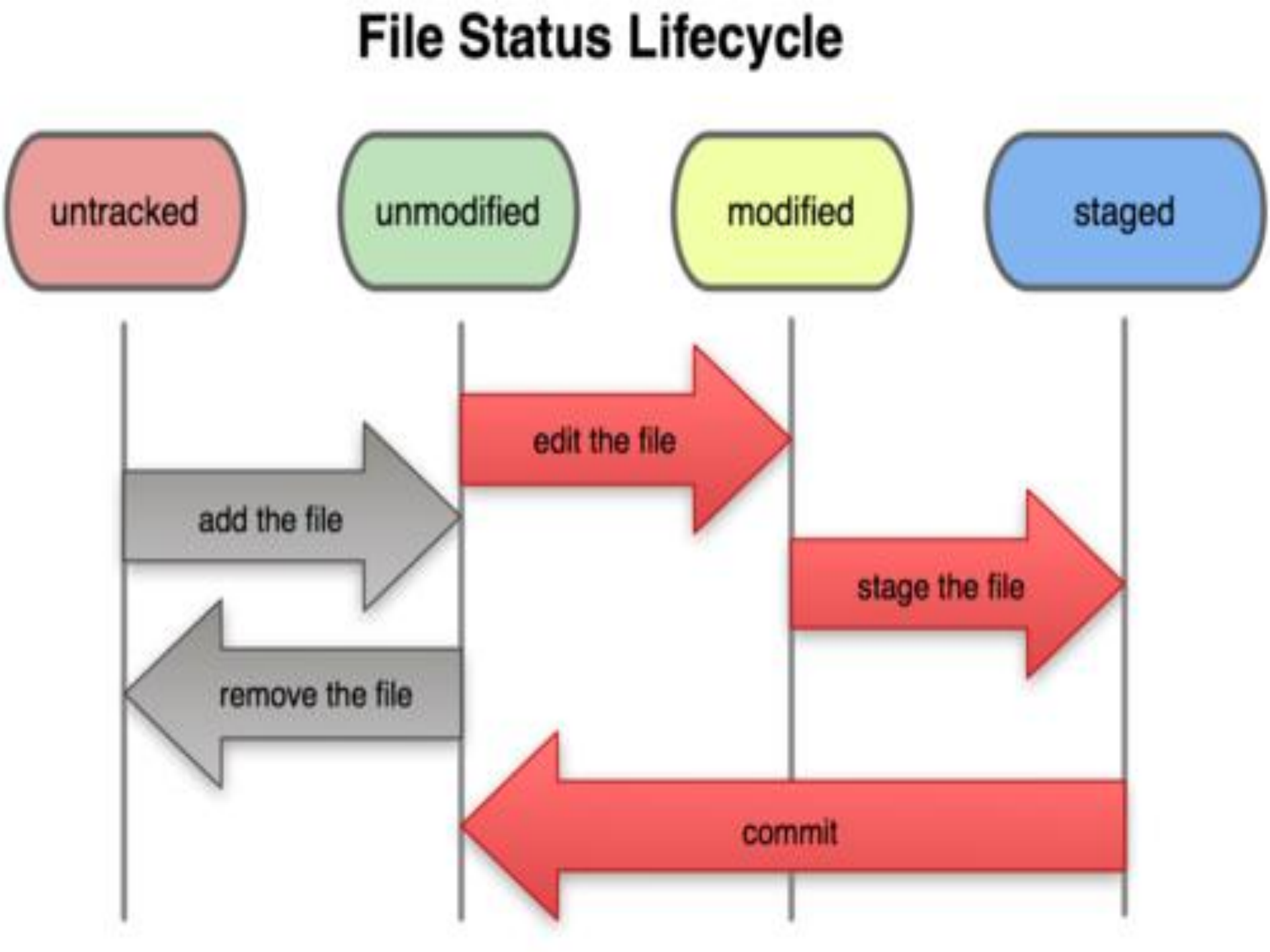
- Adding files to staging area

  git add .

  git add <file name>

- Commiting a file to Local repo

  git commit –m ‘<message>’


File Status Lifecycle
untracked
unmodified
modified
staged
edit the file
add the file
stage the file
remove the file
commit

Git commands

- Checking status of file:

  git status
- Tracking new files:

  git add <file name>
- Viewing staged & un-staged changes:

  git diff
- Viewing what you've staged so far:

  git diff --cached

Git commands

- Commit by skipping the Staging area:

  git commit –am ‘<message>’
- Removing files from working directory **:

  rm <file name>
- Un-track a file:

  git rm --cached <file name>
- Renaming a file:

  git mv <old name> <new name>

Git commands

- Viewing the commit history:

  git log

  -p : shows the diff introduced in each commit

  -2 : limits the output to only the last two entries

  --stat : shows abbreviated stats for each commit

  --pretty : changes the log output to formats other than the default

Git commands

- git log

  --pretty=oneline : prints each commit on a single line

  --pretty=format:”<format options>” : allows you to specify your own log output format

  e.g.

git log --pretty=format:"%h - %an, %ar : %s"

Format options

- %H Commit hash
- %h Abbreviated commit hash
- %T Tree hash
- %t Abbreviated tree hash
- %P Parent hashes
- %p Abbreviated parent hashes
- %an Author name
- %ae Author e-mail
- %ad Author date (format respects the --date= option)
- %ar Author date, relative
- %cn Committer name
- %ce Committer email
- %cd Committer date
- %cr Committer date, relative
- %s Subject

Git commands

- git log

  --graph

e.g.

git log --pretty=format:"%h %s" --graph

Git commands

- Using a GUI to visualize commit history:

  gitk

- Limiting log output:

  git log --since=2.weeks

Log output limiting options

- -<n> : Show only the last n commits
- --since, --after : Limit the commits to those made after the specified date.
- --until, --before : Limit the commits to those made before the specified date.
- --author : Only show commits in which the author entry matches the specified string.
- --committer : Only show commits in which the committer entry matches the specified string.

Git commands

- Changing the last commit:

```
git commit –amend
```

e.g.

```
git commit -m 'initial commit'
git add <forgotten_file>
git commit --amend
```

Git commands

- Un-staging a staged file:

git reset HEAD <file name>

- Un-modifying a modified file:

git checkout -- <file name>

Git commands

- Adding remote repo(make a connection):

git remote add <short-name> <url>

- Pushing files to remote:

git push <remote-name> <branch-name>

e.g.

git push assn master

Git commands

- Cloning from remote repo

  git clone <url>

- Fetching from remote repo:

  git fetch <remote repo>

- Pulling from remote repo:

  git pull <remote repo>

Git commands

- Showing all the remotes by short name:

  git remote

- Showing all the remotes by URL:

  git remote -v

Git commands

- Inspecting a remote:

  git remote show <remote name>

- Renaming a remote:

  git remote rename <old name> <new name>

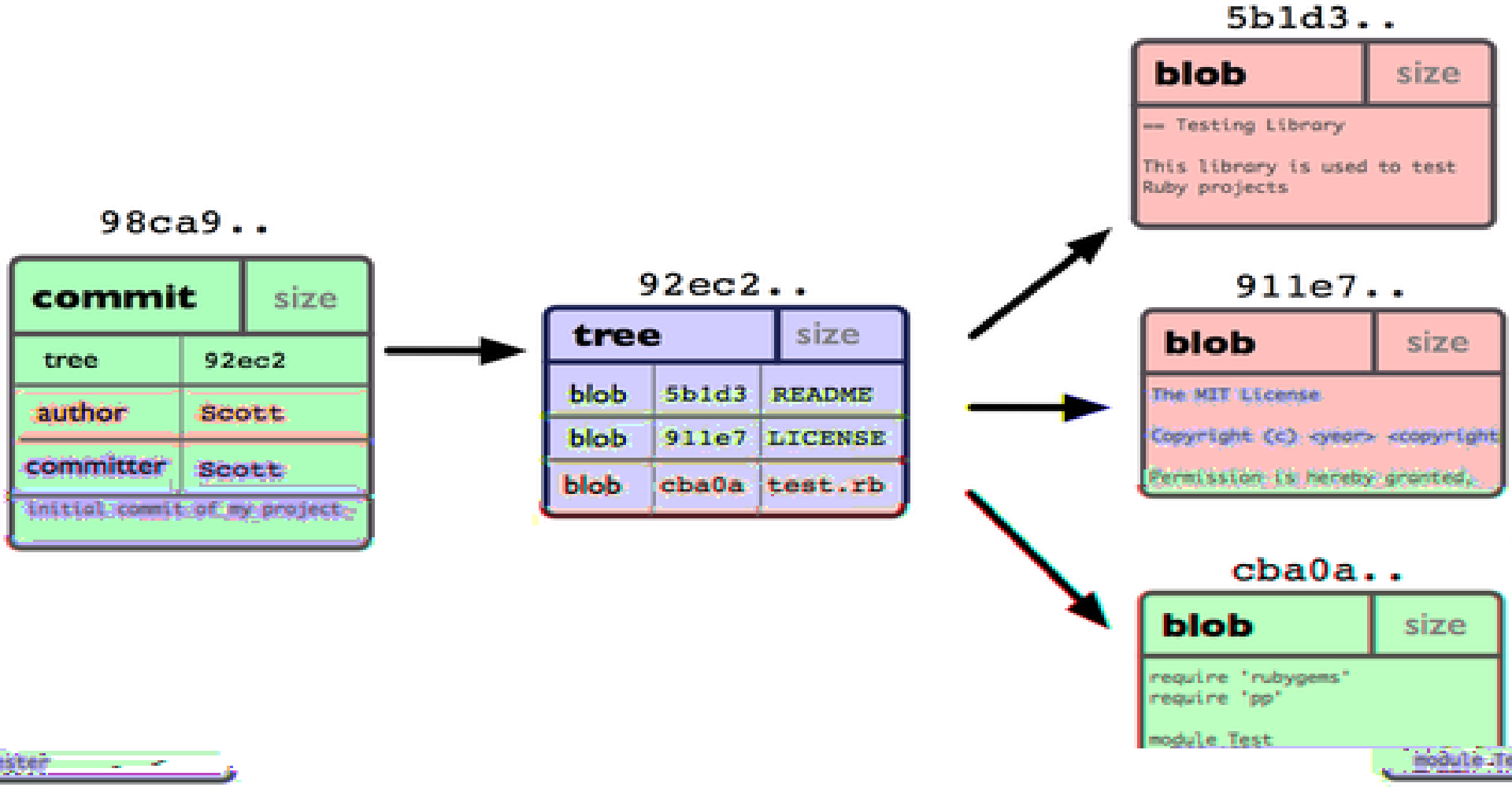
- Removing a remote:

  git remote rm <short-name>

Git Aliases

- git config --global alias.co checkout
- git config --global alias.br branch
- git config --global alias.ci commit
- git config --global alias.st status
- git config --global alias.unstage 'reset HEAD --‘
- git config --global alias.last 'log -1 HEAD'

Commit mechanism



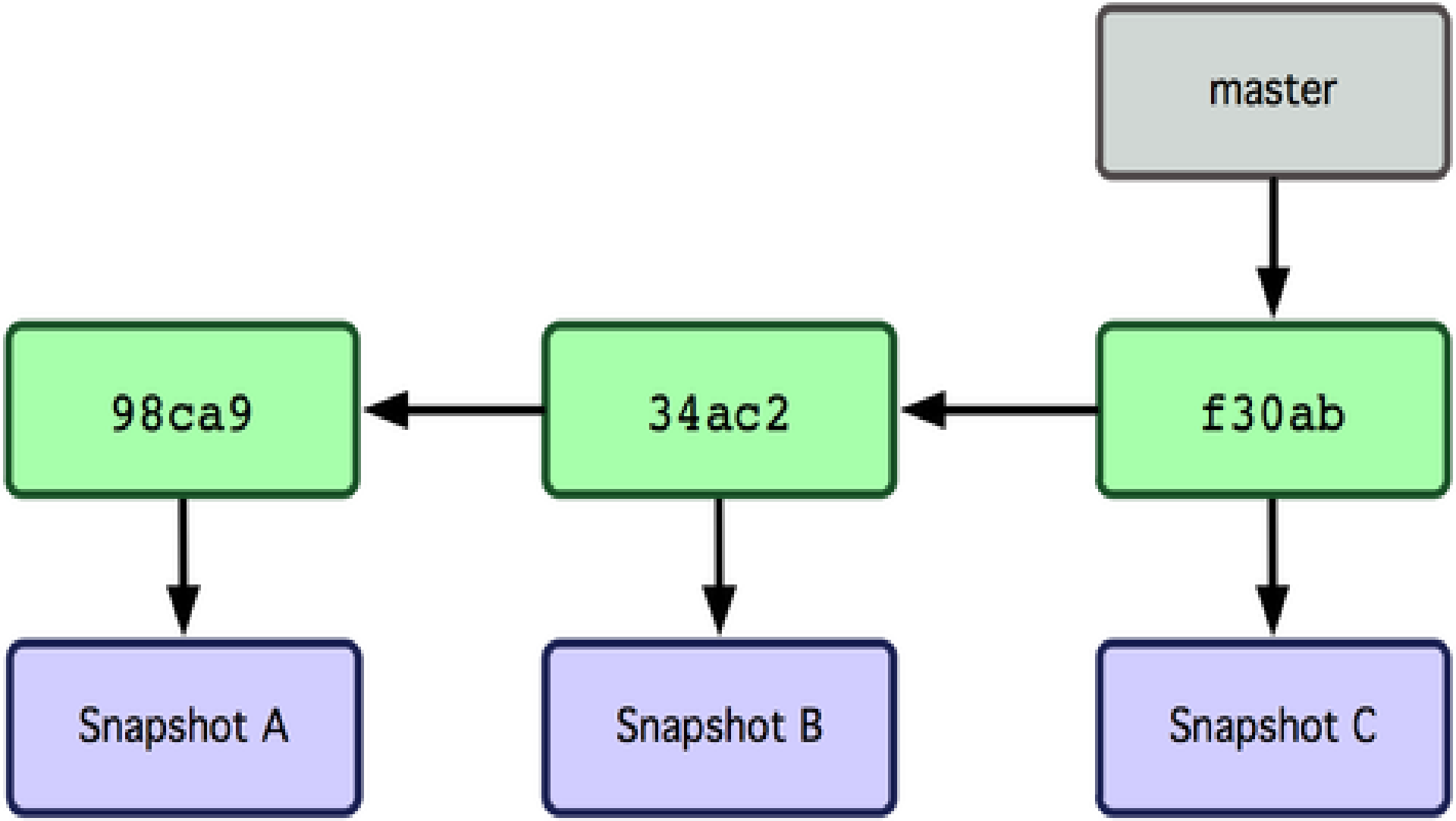
Commit mechanism

98ca9..

34ac2..

f30ab..

Branching

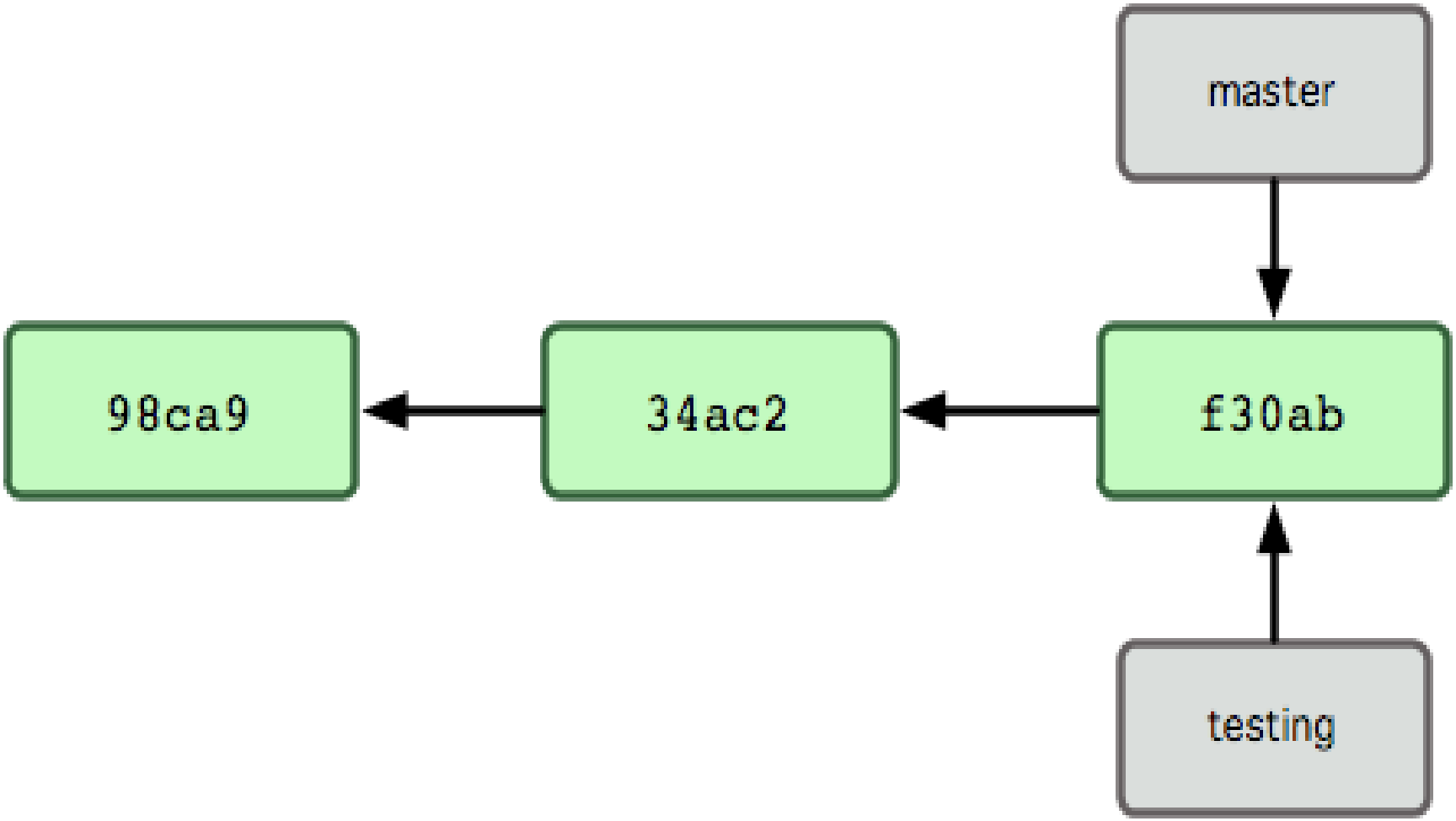


Branching

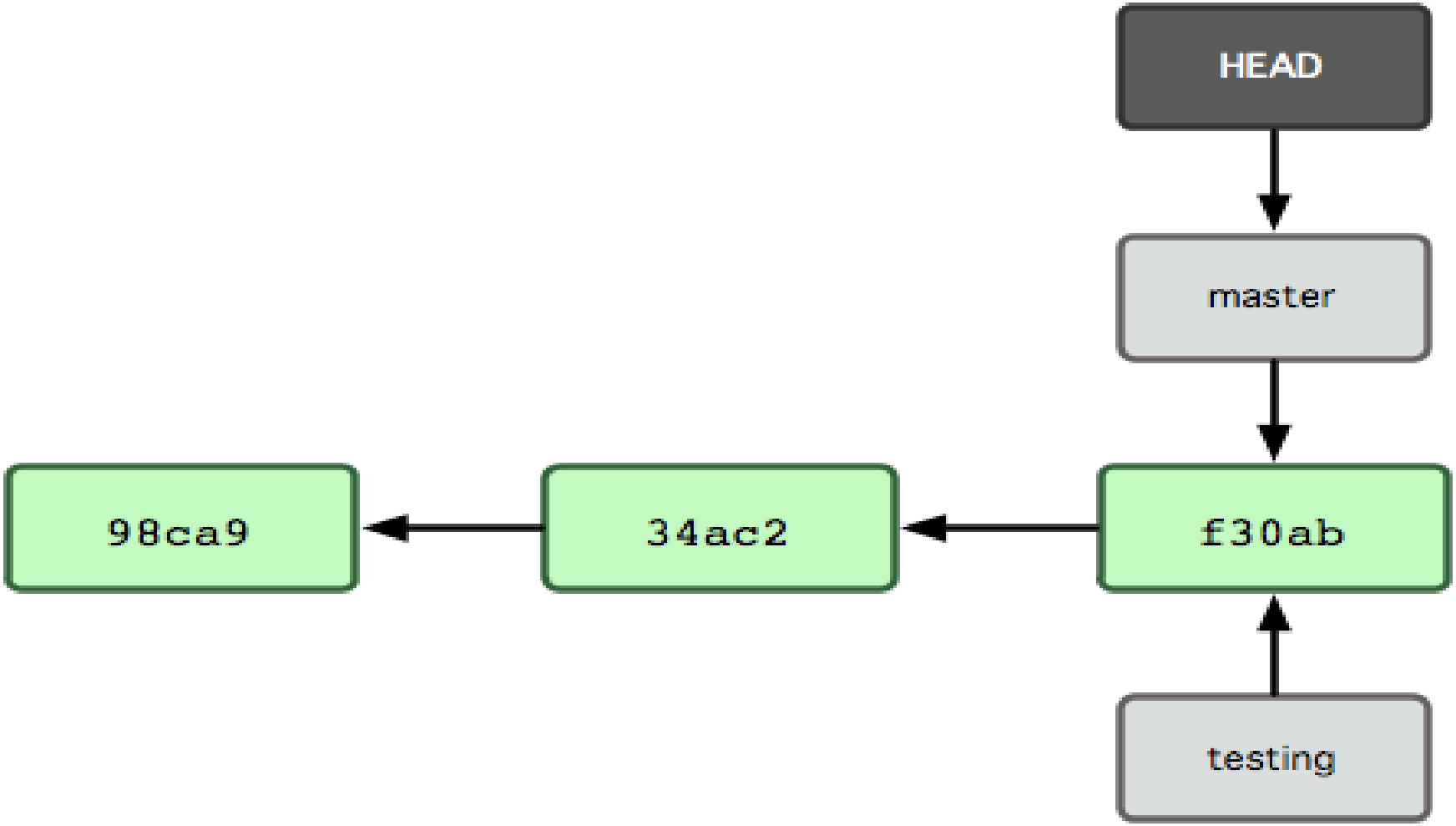
git branch <branch name>

git branch testing

Branching



Branching



Branching

- Switch to a new branch:

  git checkout <branch name>

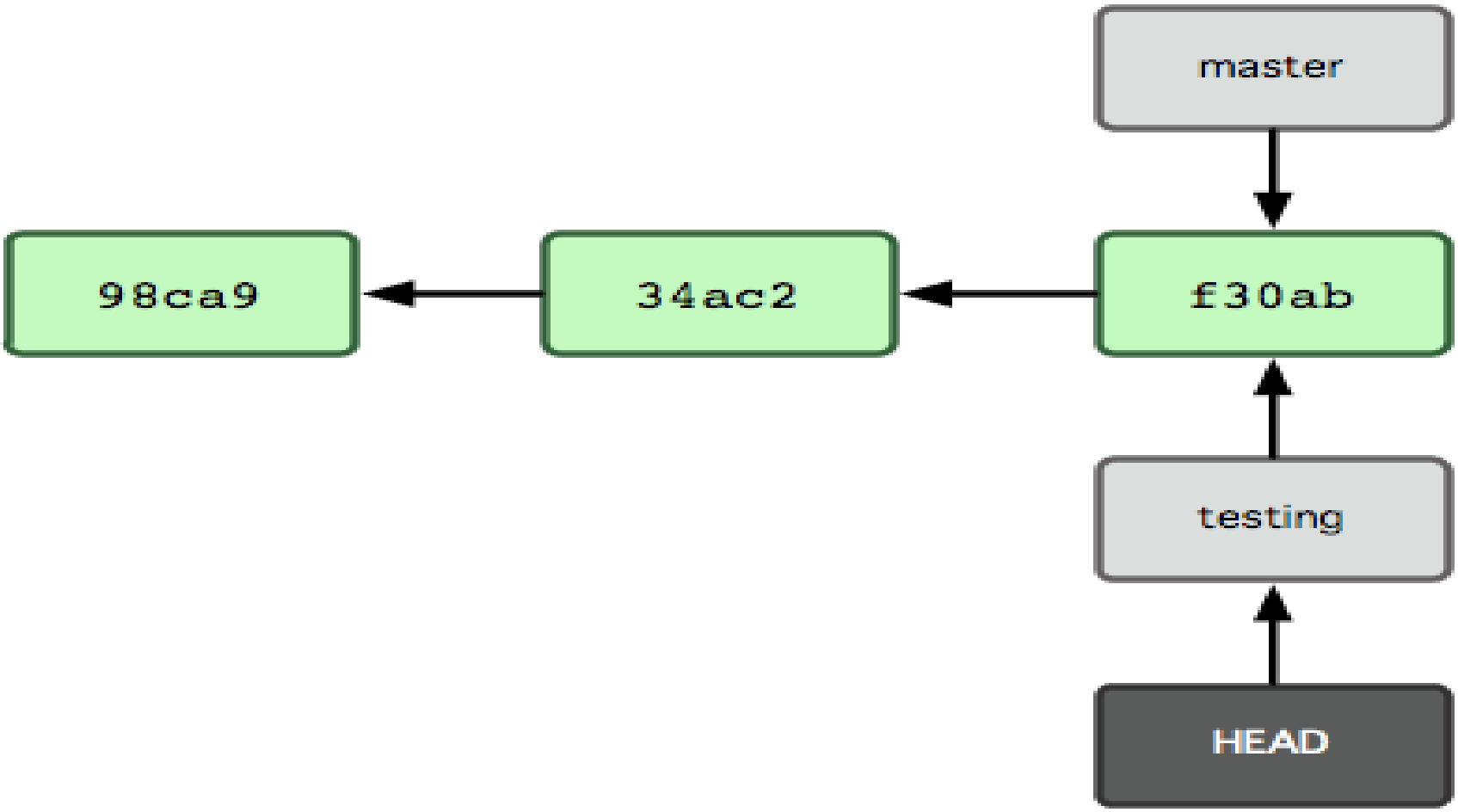
  git checkout testing

- Create & switch to a new branch at the same time:

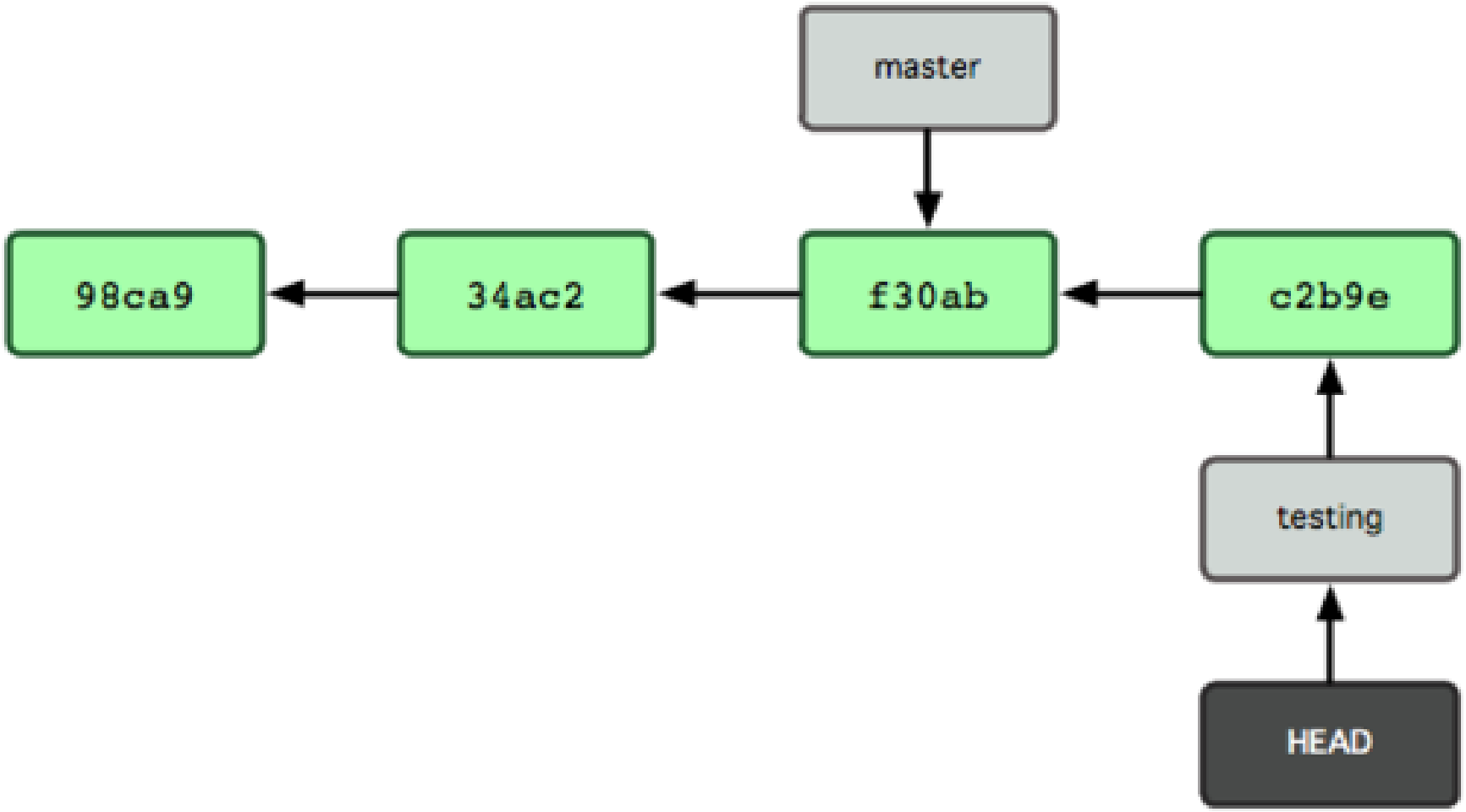
  git checkout –b <branch name>

  git checkout –b testing

Branching



Branching

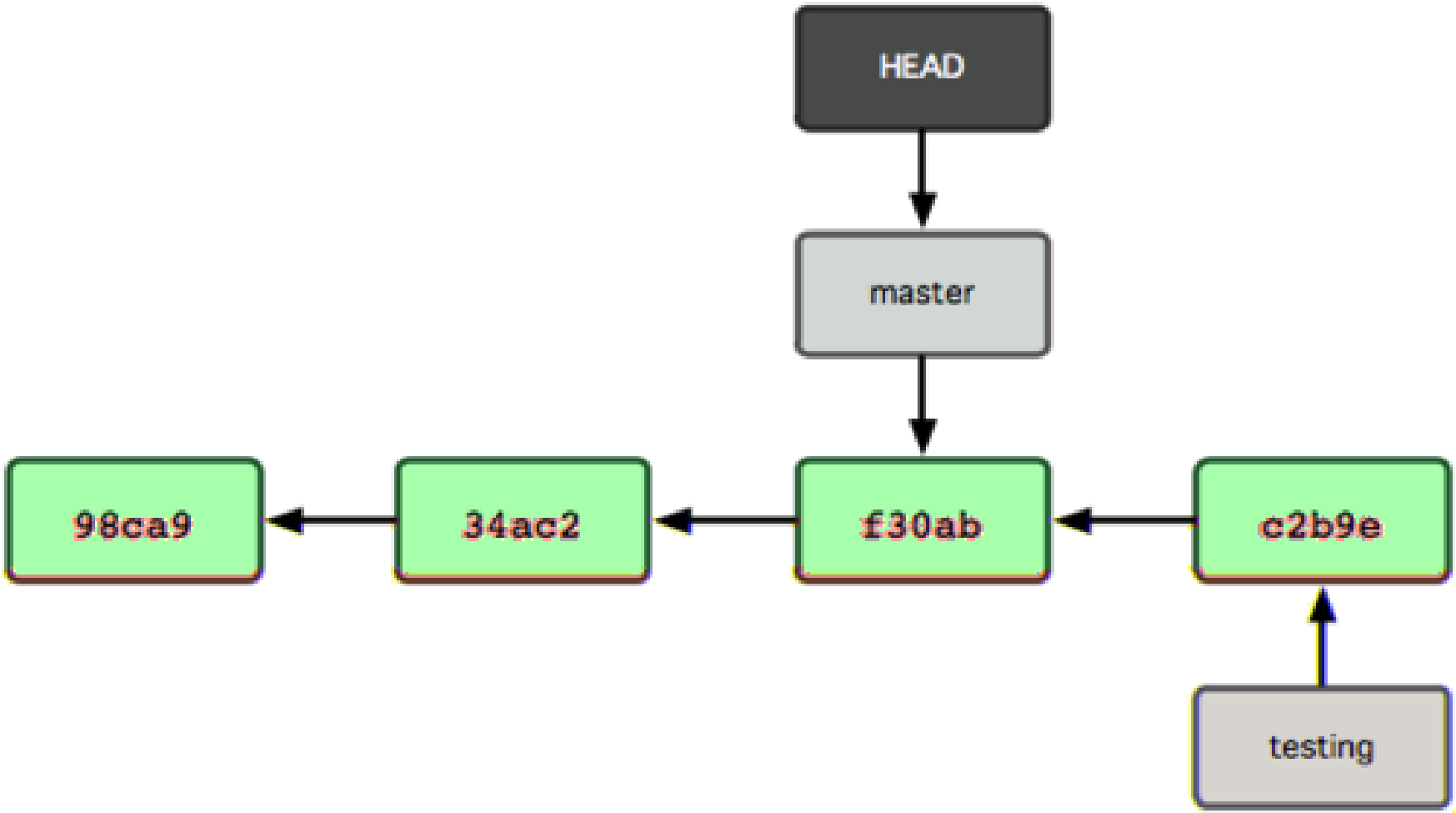


Branching

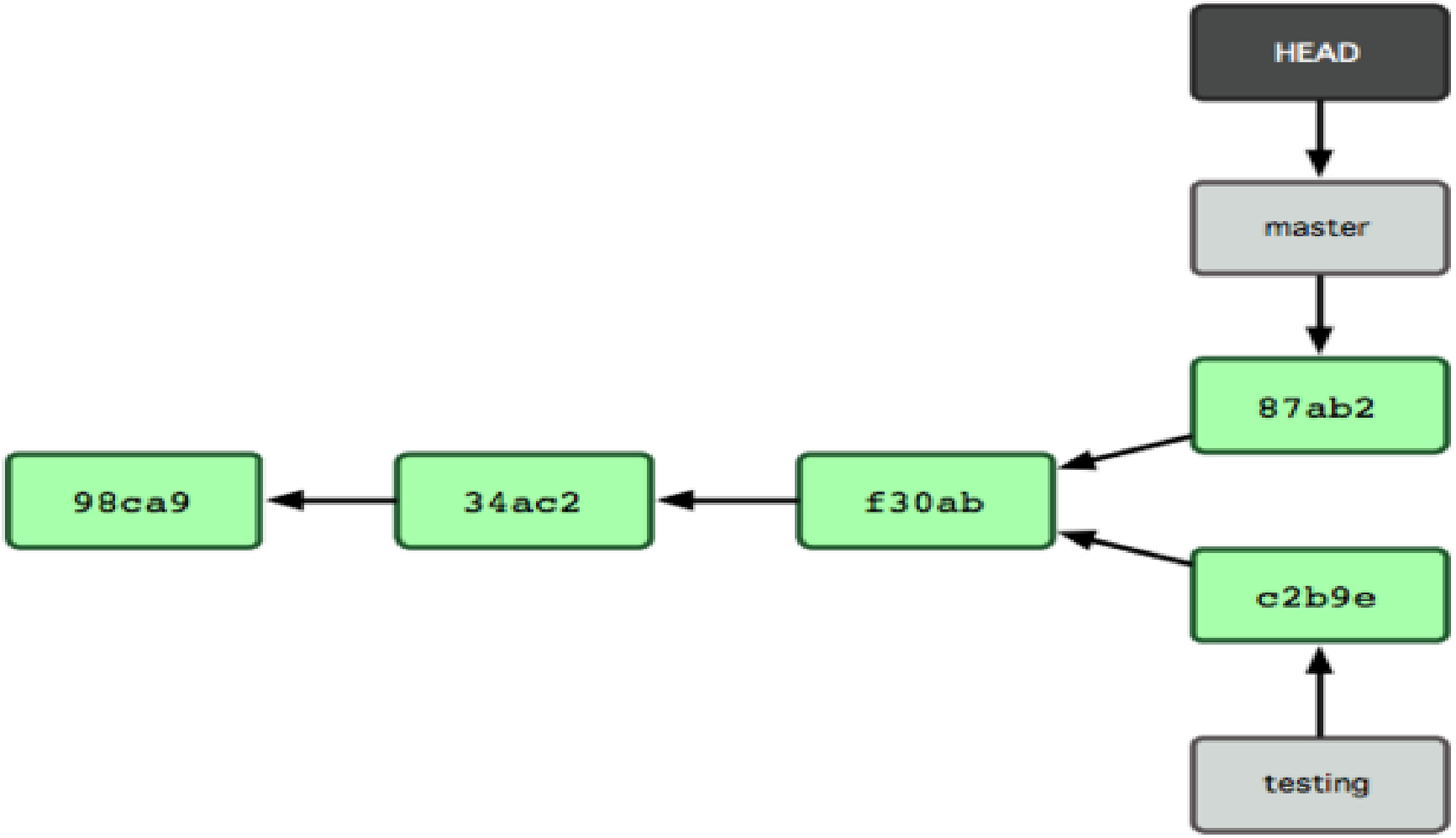
- Switch to the master branch:

  git checkout master

Branching



Branching



Branching

- Merge command:

  git checkout master

  git merge testing

- Delete a branch:

  git branch –d testing

Branching

- Show all the branches:

  git branch
- To see the last commit on each branch:

  git branch –v
- To see which branches are already merged into the branch you're on:

  git branch --merged
- To see all the branches that contain work you haven't yet merged in:

  git branch --no-merged

EGit install

- Help → Eclipse Marketplace → search for “EGit – Git Team Provider” & “GitHub Mylyn Connector”

EGit configuration

- Set up user name, email, editor:

  Window -> Preferences -> Team -> Git -> Configuration

- Set up default folder for storing Git repositories:

  Window → Preferences → Git → Team → Default Repository Folder

EGit configuration

- Activate Git Repositories perspective:

Window → Show View → Git Repositories

- Activating the git toolbar:

Window → Customize perspective →
Command Groups Availability → check Git &
Git Navigation Actions

EGit

- Git init:

  Right click on the project → Team →

  Share Project → Git

Ignoring files

- .gitignore file
- Right click → Team → Ignore
- Ignore derived resources, e.g. class files: Window → Preferences → Team → Git → Projects → Automatically ignore derived resources

EGit

- Open Git staging view:

  Window → Show View → Other→ Git → Git Staging

- Stage
- Commit

EGit

- Git log:

  Right click → Team → Show in history

- If you want to see more details about a commit: right-click → Open in Commit Viewer

EGit

- Add remote:

  Git Repositories perspective → expand the local repo → right-click on Remote → Create Remote

- Push

EGit

- Git clone:

  File → Import → Git → Projects from Git -- > URI → paste url (from GitHub) → Authentication

DONE!!!!!

THANK YOU !!!!!!!!!!