GIT BASICS

CLARA BENNETT

WHAT IS GIT?

GIT IS A DISTRIBUTED VERSION CONTROL SYSTEM

» Preserves change history

» Allows splitting and merging together of branches

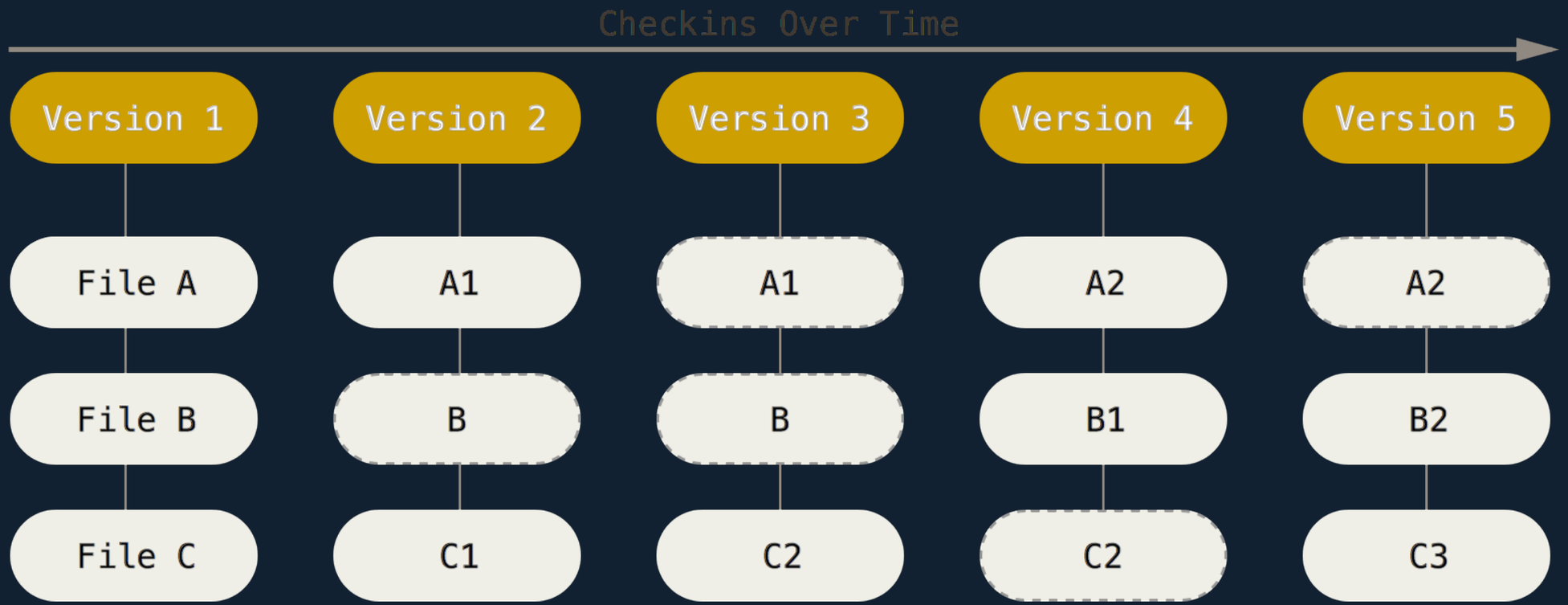
» All repository copies are equal

» (almost) All operations are local

HOW DOES IT WORK?

SNAPSHOTS

» Check in changes → Git saves working directory snapshot



SHA-1 CHECKSUMS

» Commit snapshots are check-summed with SHA-1

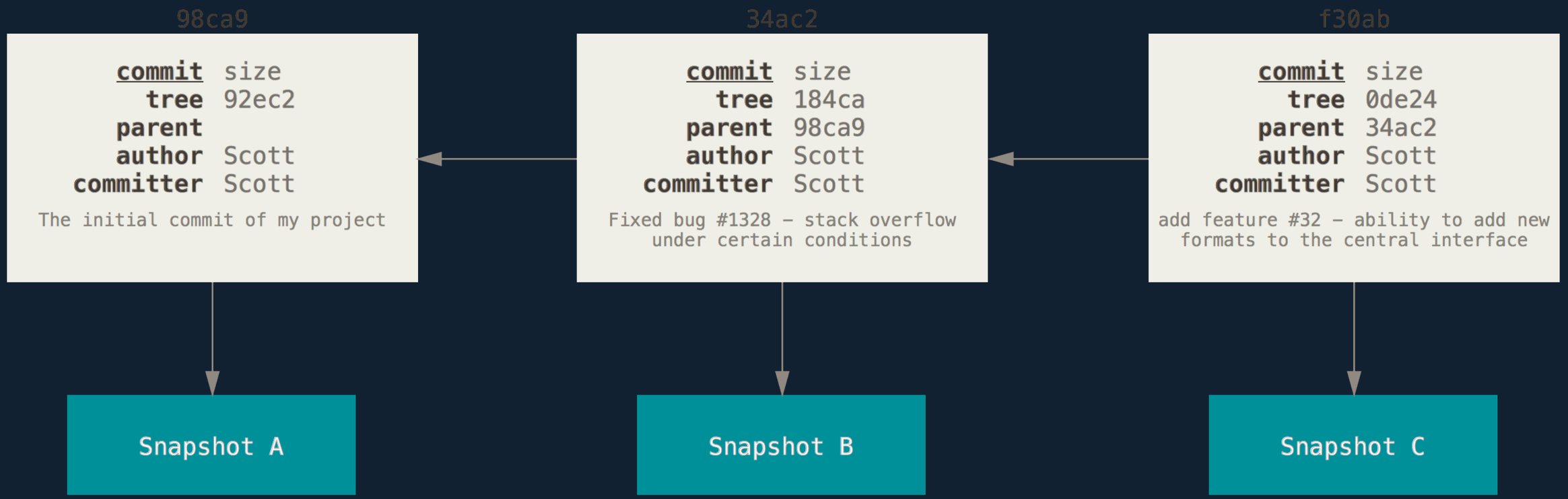
» The resulting hash serves to identify the commit

» Changes are detected by comparing checksums

» Note: any modification of the change history (including to commit messages) results in new commit identities

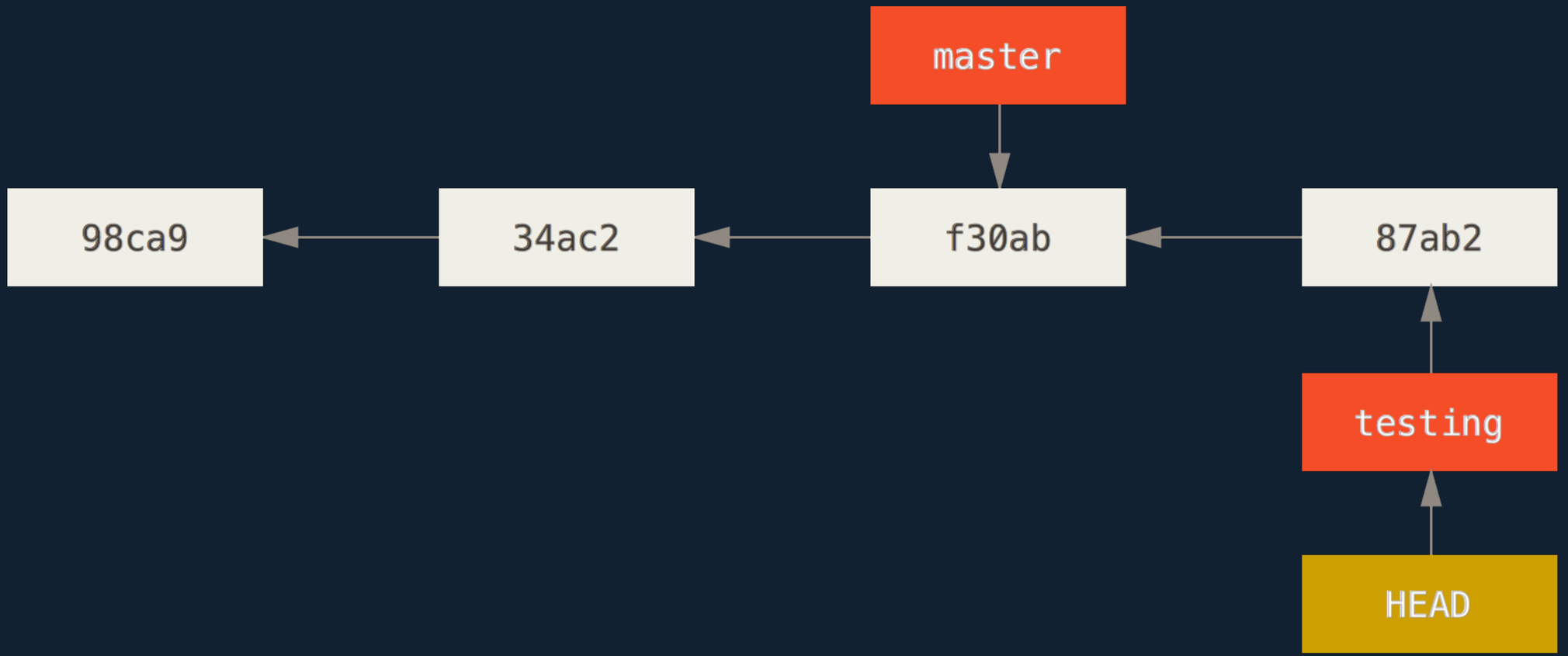
COMMIT TREES

» Commits know who their parents are



BRANCHES

» A branch is a named pointer at a commit



SETTING UP A REPO

NEW LOCAL REPOSITORY

» For a brand-new repository

```
$ mkdir git_basics
$ cd git_basics
$ git init
```

» For a local copy of an existing repository

```
$ git clone git@github.com:csojinb/git_basics.git
```

IGNORING FILES

» We only want to version-control meaningful changes

» Repo-specific generated files (e.g. *.pyc, *.log) should be ignored in the version-controlled .gitignore

» Files generated by your specific workflow (e.g. editor files, ipynb_checkpoints/) should be ignored at the user level

```
$ git config --global core.excludesfile ~/.gitignore_global
```

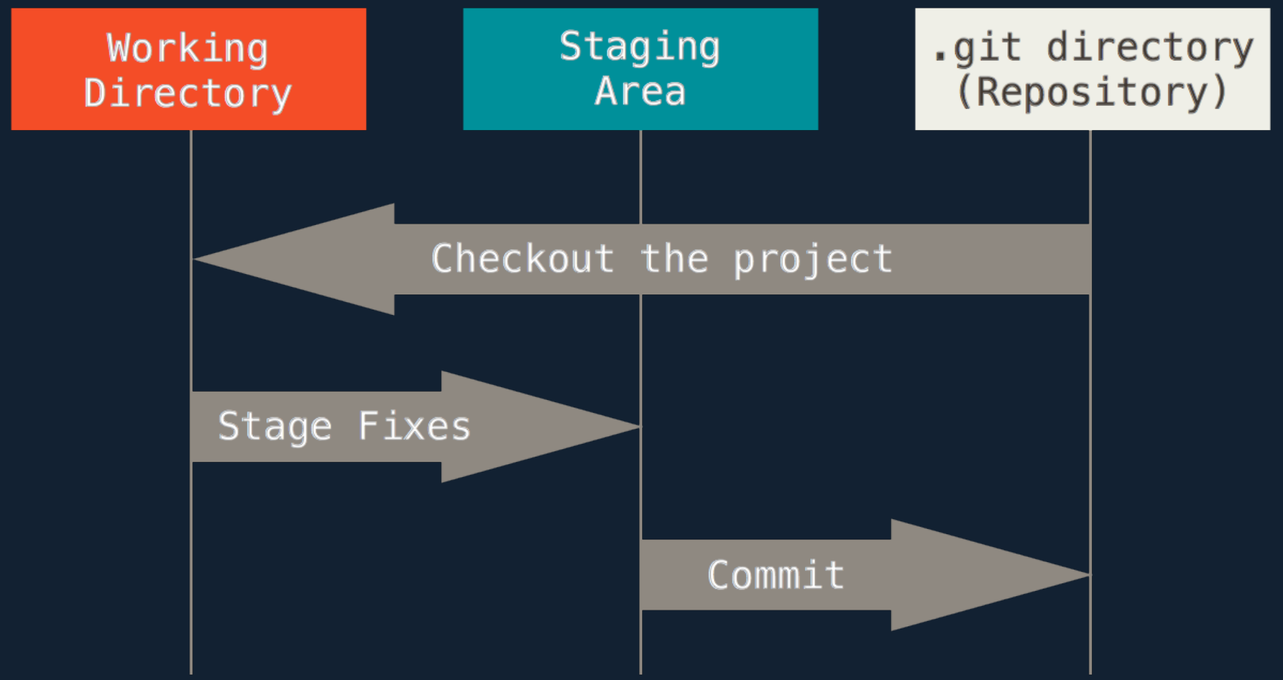
COMMITTING CHANGES

PROJECT AREAS

» Working directory: untracked files, unstaged modifications

» Staging area: changes flagged for committing

» .git directory: committed change history



MOVE FILES TO/FROM STAGING

» Use git add to stage untracked or modified files

```
$ git add <filename>
$ git add --all [dir]
```

» To unstage: git reset HEAD[--<filename>]

» To untrack: git rm --cached

» Note: bare git rm will delete the file altogether

COMMIT WITH A GOOD MESSAGE

» Commit logically grouped sets of changes together

» Run git status before committing, to verify

» Run git commit and resist the urge to use the -m flag

» Write a nice, descriptive message in the editor → consider using multiple lines to explain!

BRANCHING & MERGING

CREATE AND NAVIGATE BRANCHES

» To check out a branch: git checkout branch_name

» To check out a commit: git checkout <SHA-1>

» Create a new branch: git branch new_branch

» To create and check out a new branch:
$ git checkout -b new_branch

MERGING

- Always work on a branch
  - Branches are cheap (just pointers)
  - Digging out code you don't want anymore is hard
- When your branch is ready, merge it into master

```
$ git checkout master
$ git merge branch_name
```

PULLING & PUSHING

REMOTE REPOSITORIES

- Remotes are other people's copies of the same repository
  - If you clone a repo, the original is automatically added as the remote named origin
- Any other copy accessible by https or ssh can be added as a remote → even others' local repos

```
$ git remote add ani git@github.com/anivemprala/git_basics.git
```

PULLING

» Update local cache of remote repository:

```
$ git fetch remote_name
```

» Fetch and merge into checked-out branch:

```
$ git pull [remote_name branch_name]
```

» Reset to remote branch, discarding local changes

```
$ git reset --hard remote_name/branch_name
```

PUSHING

» Push changes to a same-named remote branch:
  $ git push remote_name branch_name

» Push changes to a differently-named remote branch:
  $ git push remote_name local_branch:remote_branch

» Note that Github will prevent pushes that will overwrite existing changes; if your push is rejected, you'll need to pull down the remote changes first

TRACKING BRANCHES

» Local branches can "track" remote branches

  » In a cloned repo, master automatically tracks origin/master

» git pull or git push default to tracked branch

» Set tracking branch:
  $ git branch -u remote_name/branch_to_track

» Set and push to tracking branch:
  $ git push -u remote_name branch_to_track

THE END