



git

Outline

- Introduction to Version Control Systems
- Origins of Git
- Git terminology & concepts
- Basic Git commands
- Branches in Git
- Git over the network

Why do I need version control?

- How many lines of code was the biggest project you've worked on thus far?
- Were you collaborating with others on that project?
- How did you share code?
- How did you keep track of different changes?
- Were you ever in a place where the project once worked but it didn't when you needed it?

What does VCS do for me?

- Wikipedia: In computer software engineering, revision control is any kind of practice that tracks and provides control over changes to source code.

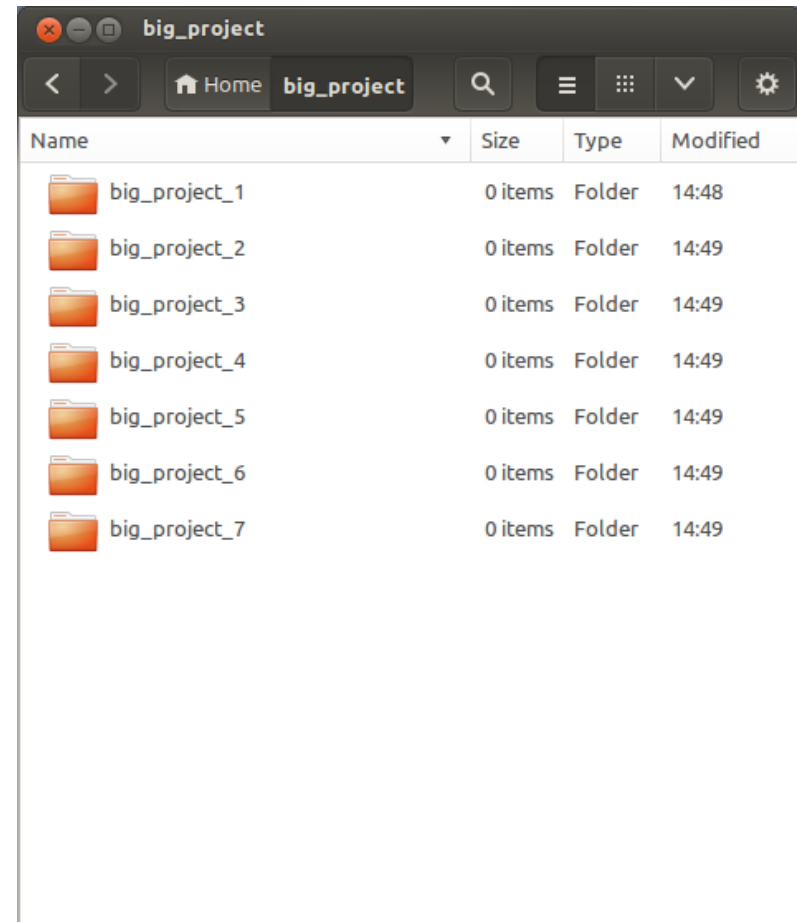
What does VCS do for me?

What we want from a VCS:

- Save a history of changes we make
 - Visibly show the history
 - Allow us to revert to an older state
- Aid in collaboration with others
 - Should include tools for combining code together
- Aid in creating backups of our work

The local approach

- No special program, just a collection of folders
- Versions identified by a number or date
- By early 1980's developers started using databases to store deltas (rcs)

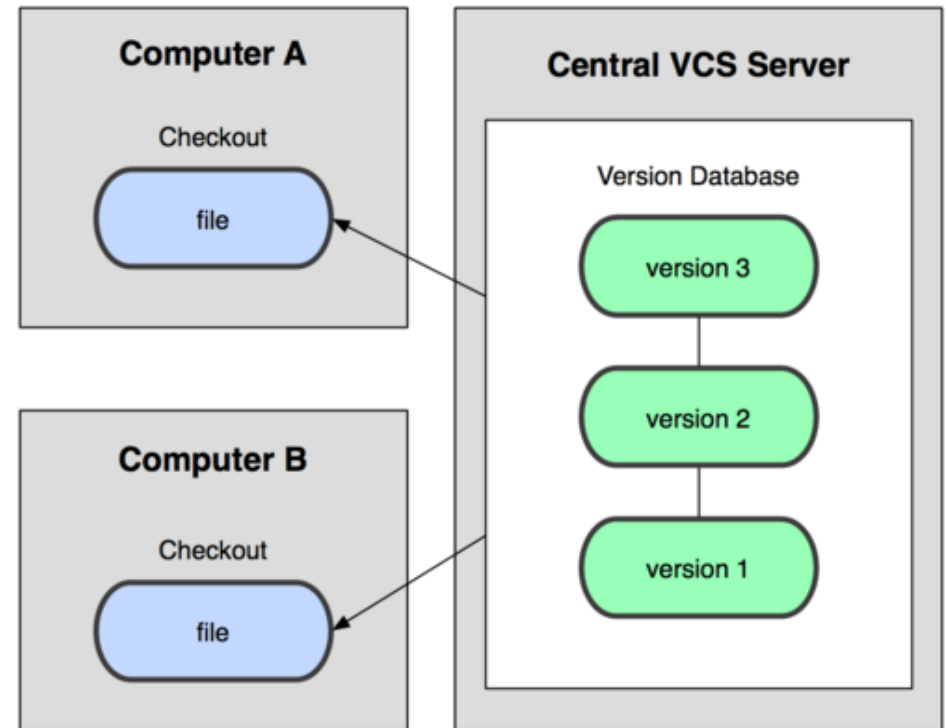


AINT NOBODY GOT TIME

FO DAT

The centralized approach

- Server-Client model
- A single server stores all revisions
- Clients check-out the revision they want



The centralized approach

Pros:

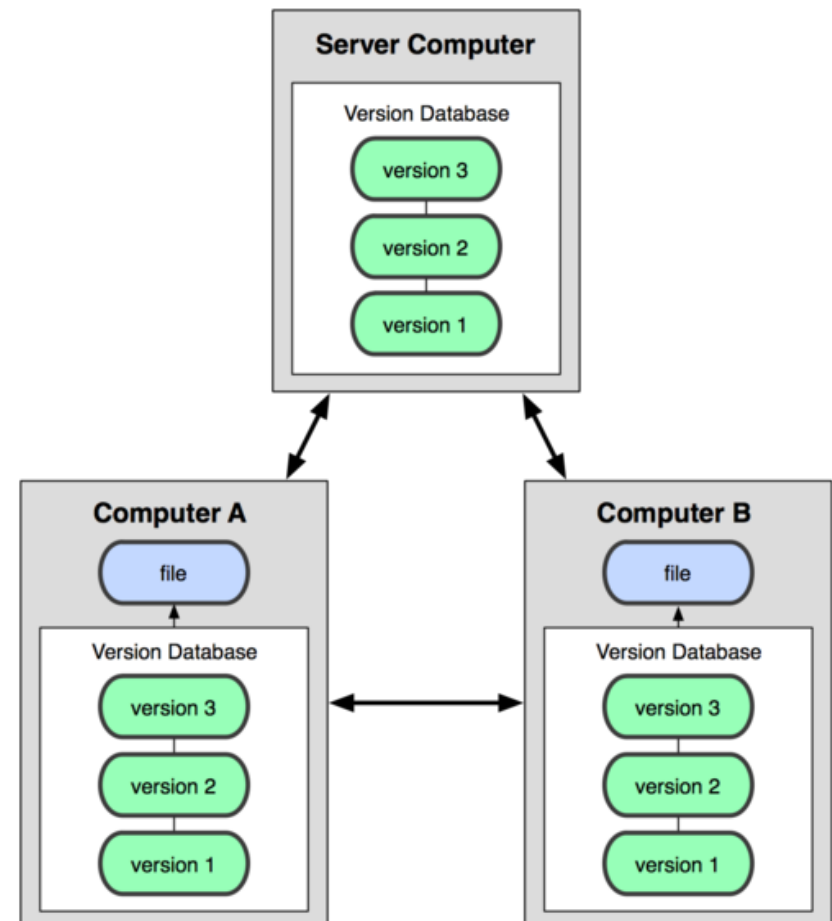
- Allows you to share code with teammates
- Keeps a well structured history

Cons:

- Single point of failure at the server

The distributed approach

- Everyone has the full history of the project
- Possible to have more than one server
- Crashed server is only a minor speedbump
- Easy to recover from bad hard drive on server

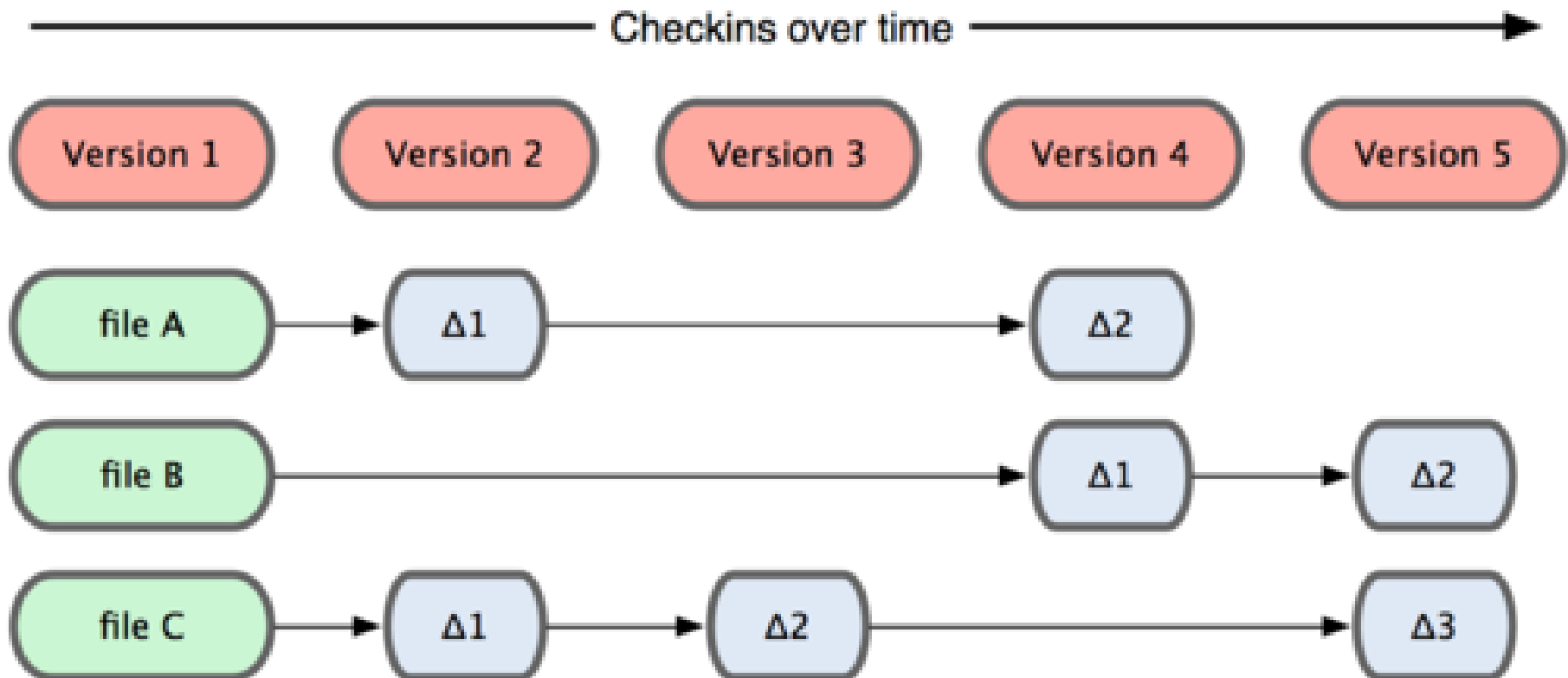


The origins of Git

- From 1991-2002 Linux was developed by emailing patch files around
- From 2002-2005 Linux used a proprietary VCS
- The copyright holder of the proprietary VCS revoked Linux's free use of the software
- Linus, and others, built their own VCS, Git.

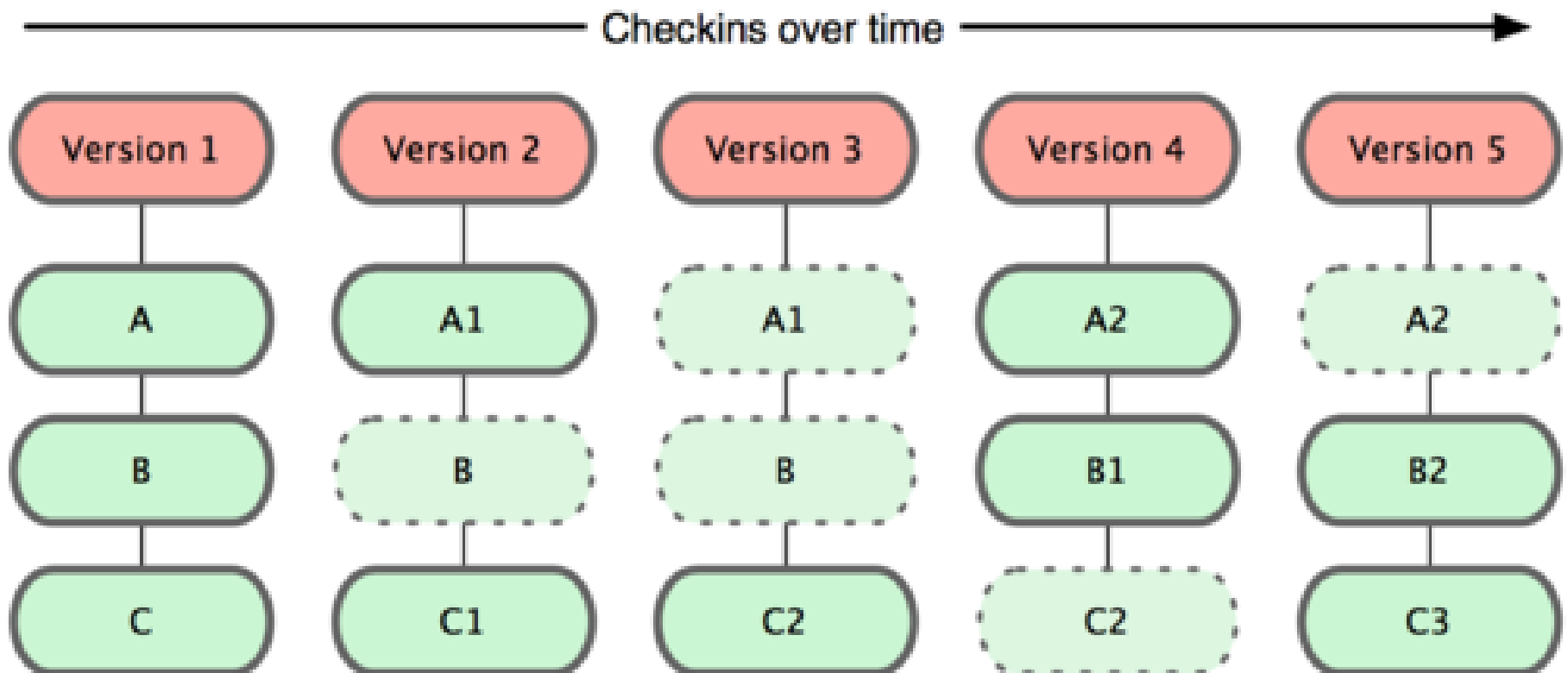
Diffs vs Snapshots

Most VCS's track project history by storing diff files.



Diffs vs Snapshots

Git tracks project history by storing snapshots of the project directory.

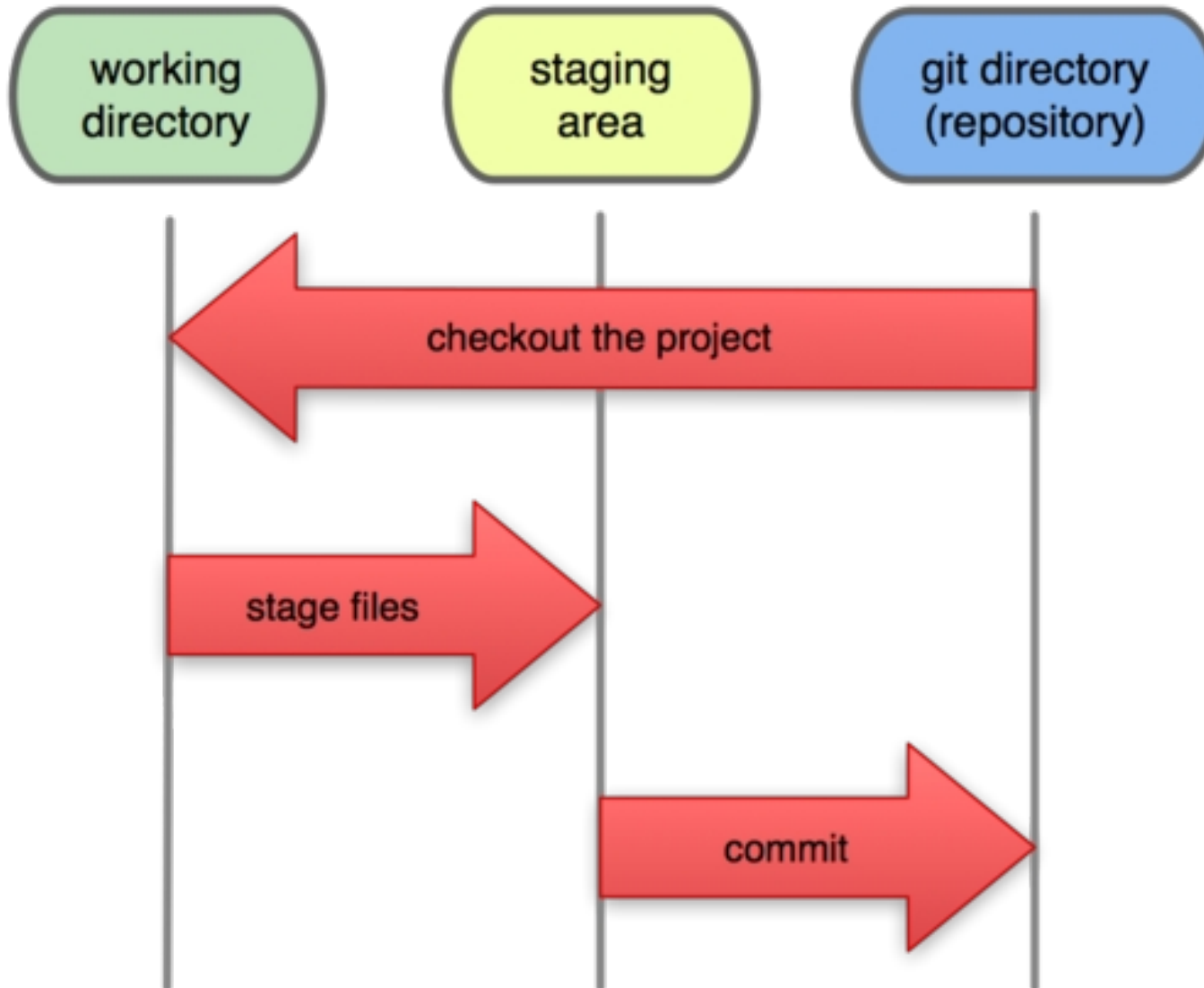


The three states of a file

- **Modified**
 - The file has been modified after you checked it out
- **Staged**
 - You tell git when to stage a file(s)
 - Files must be staged before they are committed
- **Committed**
 - You tell git when to commit a file(s)
 - Committed files are recorded in a database

The three states of a file

Local Operations



First steps

First thing you do in Git needs to be to tell Git who you are. This is needed for your work history when collaborating with others.

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email
```

```
"johndoe@aol.com"
```


Git Init

You have to tell git where you want to make a repository. This path will be the top directory of your project.

```
$ cd ~/school/ece473/lab/lab_1
```

```
$ git init
```

Git Status

The “git status” command tells you which branch you are on, which files are modified, and which are staged.

```
$ git status
```

```
# On branch master
```

```
#
```

```
# Initial commit
```

```
#
```

Git Add

After adding or modifying files we use the “git add” command to stage files.

```
$ vim main.c
```

```
$ vim fm_radio.c
```

```
$ git add *.c
```

Git Commit

The git commit command saves into the database a new snapshot of your project with the changes in the staging area (git add)

```
$ git commit
```

Important flags:

-a: add all new/modified files and then commit

-m: write commit message in command

```
$ cp ~/math_lib/ .
```

```
$ git commit -a -m "Add math library to  
project."
```

Git Log

The git log command prints to the terminal the history of your commit messages for the branch that you are currently on.

```
$ git log
```

```
Commit:
```

```
e95513e4fe173ca6b0246d2a4c85057ee41b639c
```

```
Author: Micah Losli <micah.losli@gmail.com>
```

```
Date: (7 days ago) 2013-11-11 21:22:53 -0800
```

```
Subject: Add support for song options
```

Git RM

The `git rm` command removes the given file(s) and adds the removal operation to the index (staging area).

```
$ git rm <file>
```

Git MV

The `git mv` command moves/renames files, ensures that git understands the change, and adds the change to the index.

```
$ git mv old_file.c new_file.c
```

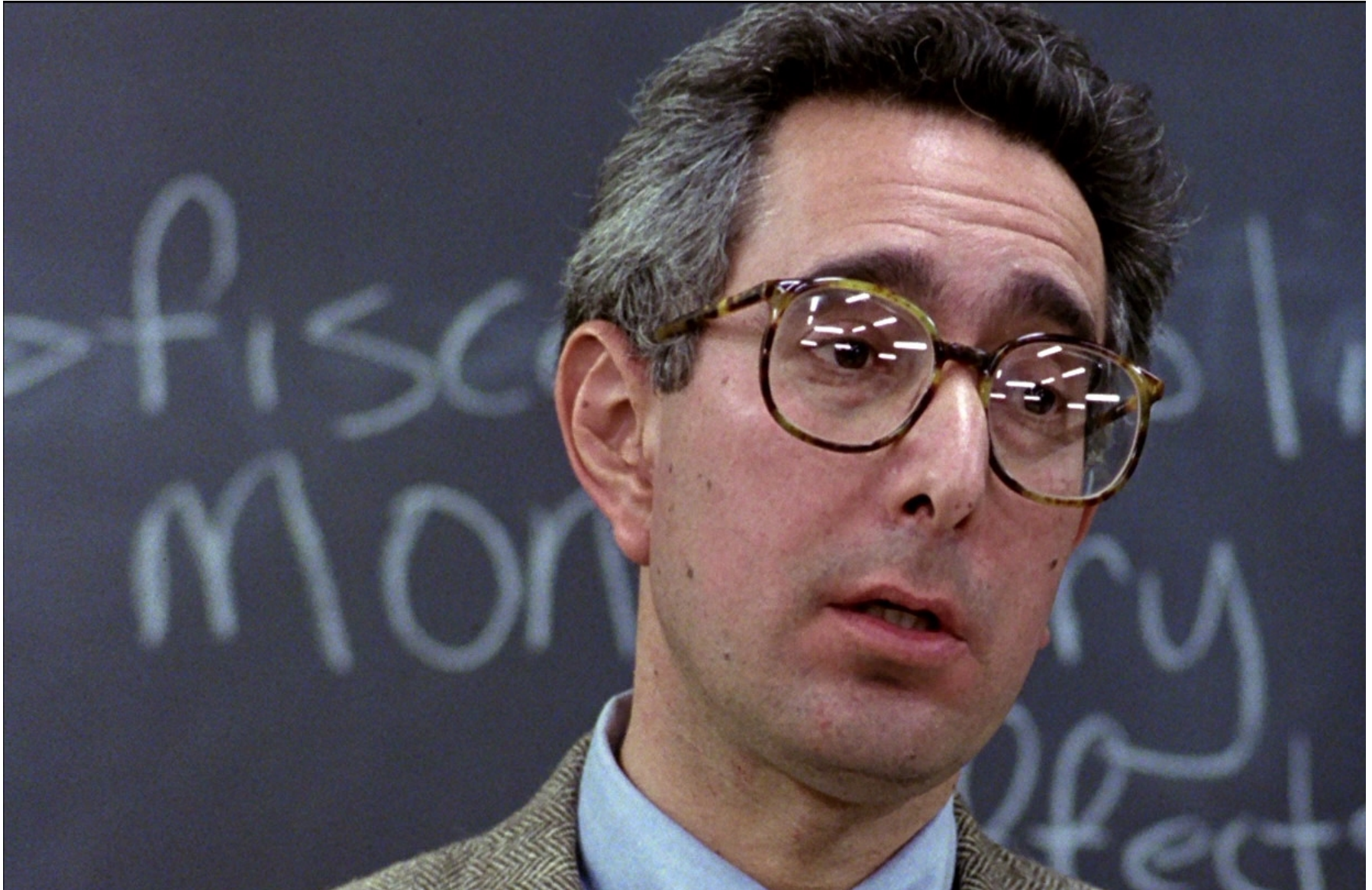
VS

```
$ mv old_file.c new_file.c
```

```
$ git rm old_file.c
```

```
$ git add new_file.c
```

Good so far?



In Class Exercise Part 1



Branch?

“Branching means you diverge from the main line of development and continue to do work without messing with the main line.”

-Scott Chacon, Pro Git

“Think of a branch as you would a copy of your project folder: a safe place to experiment with new code.”

-Micah Losli

What is a branch?

- In most VCSs, a branch is actually a new copy of your project directory. With a large project, this is very expensive in terms of disk space and processing power to make
- In Git, a branch is merely a pointer to a snapshot (aka commit). This makes working with branches in git very “cheap”.

Git Branch

```
$ git branch
```

- List local branches

```
$ git branch <branch_name>
```

- Make a new branch with name `branch_name`

```
$ git branch -d <branch_name>
```

- Delete the `<branch_name>` branch

Git Checkout

The `git checkout` command changes our current branch. When the branch is changed, the project directory is updated to the contents of that branch.

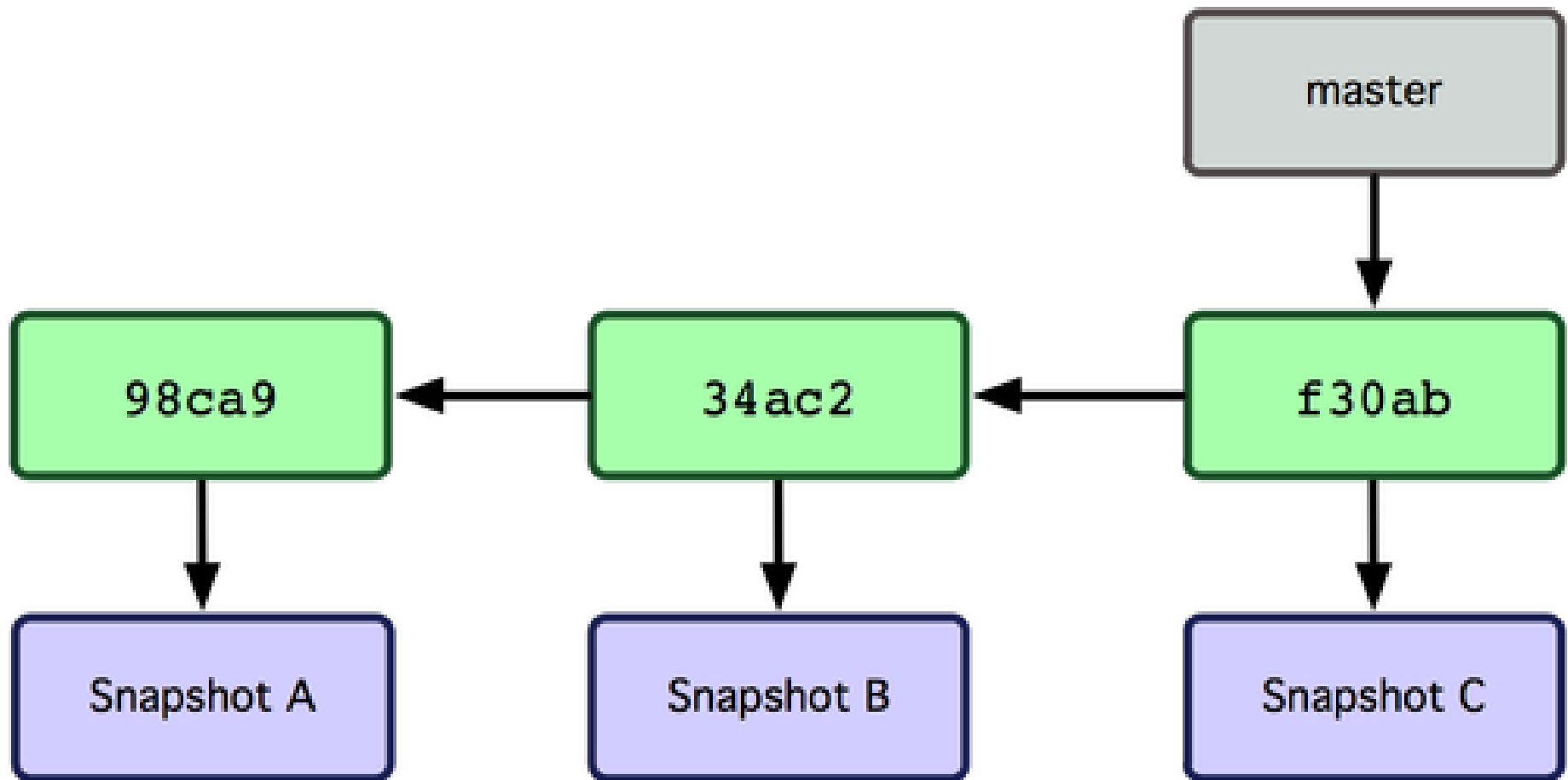
```
$ git branch testing
```

```
$ git checkout testing
```

Or you can use an equivalent shortcut:

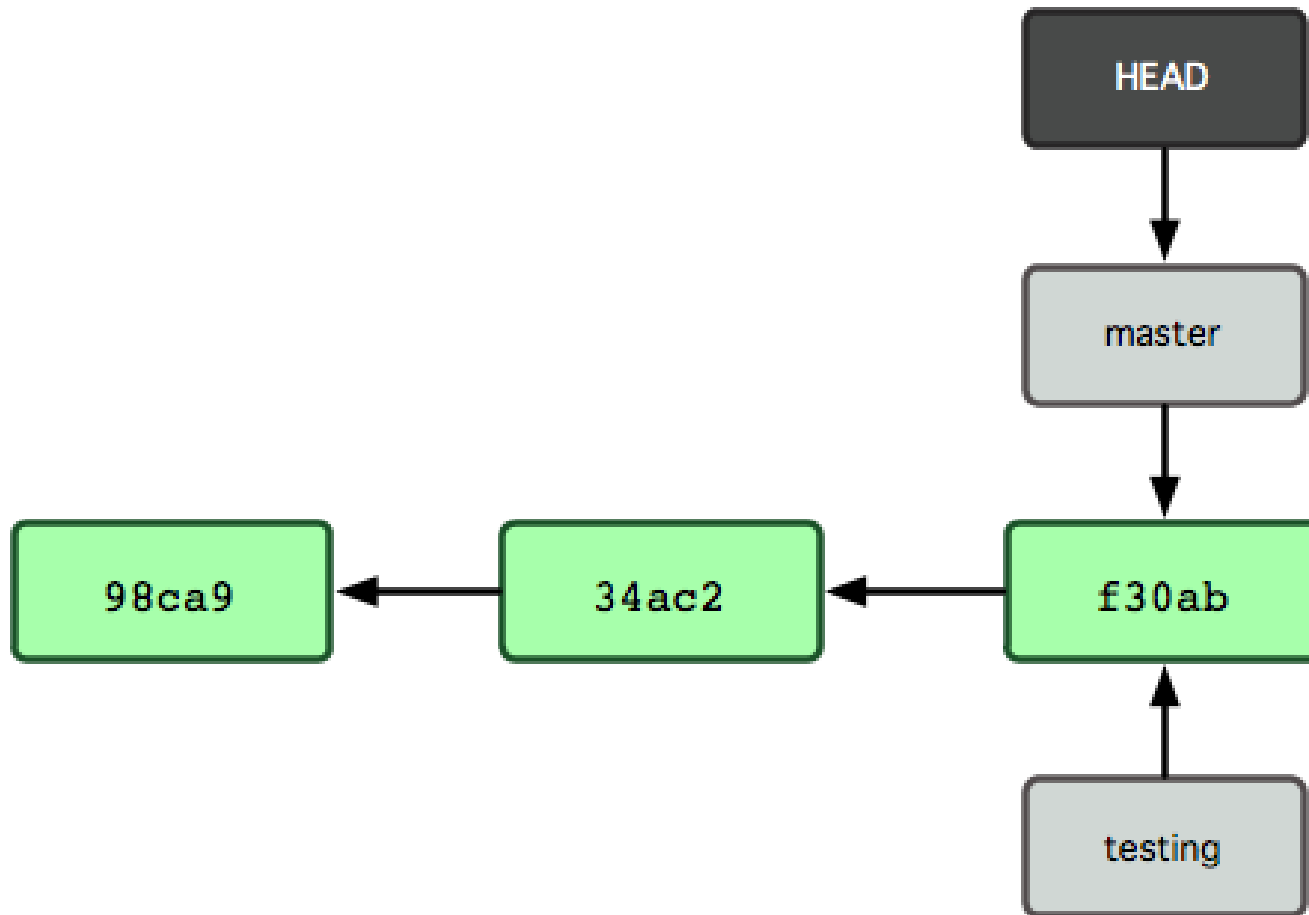
```
$ git checkout -b testing
```

Branches are pointers



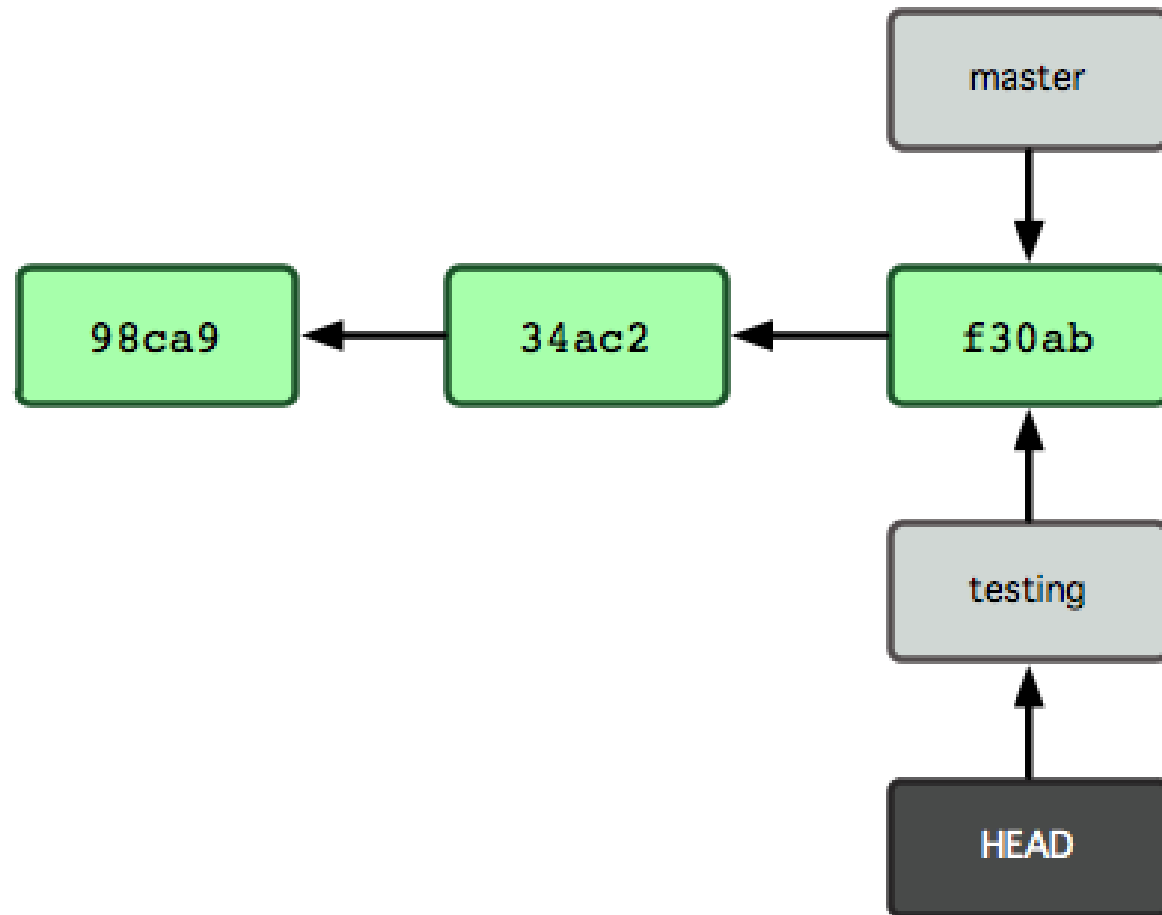
A repository after 3 commits and no new branches.

Branches are pointers



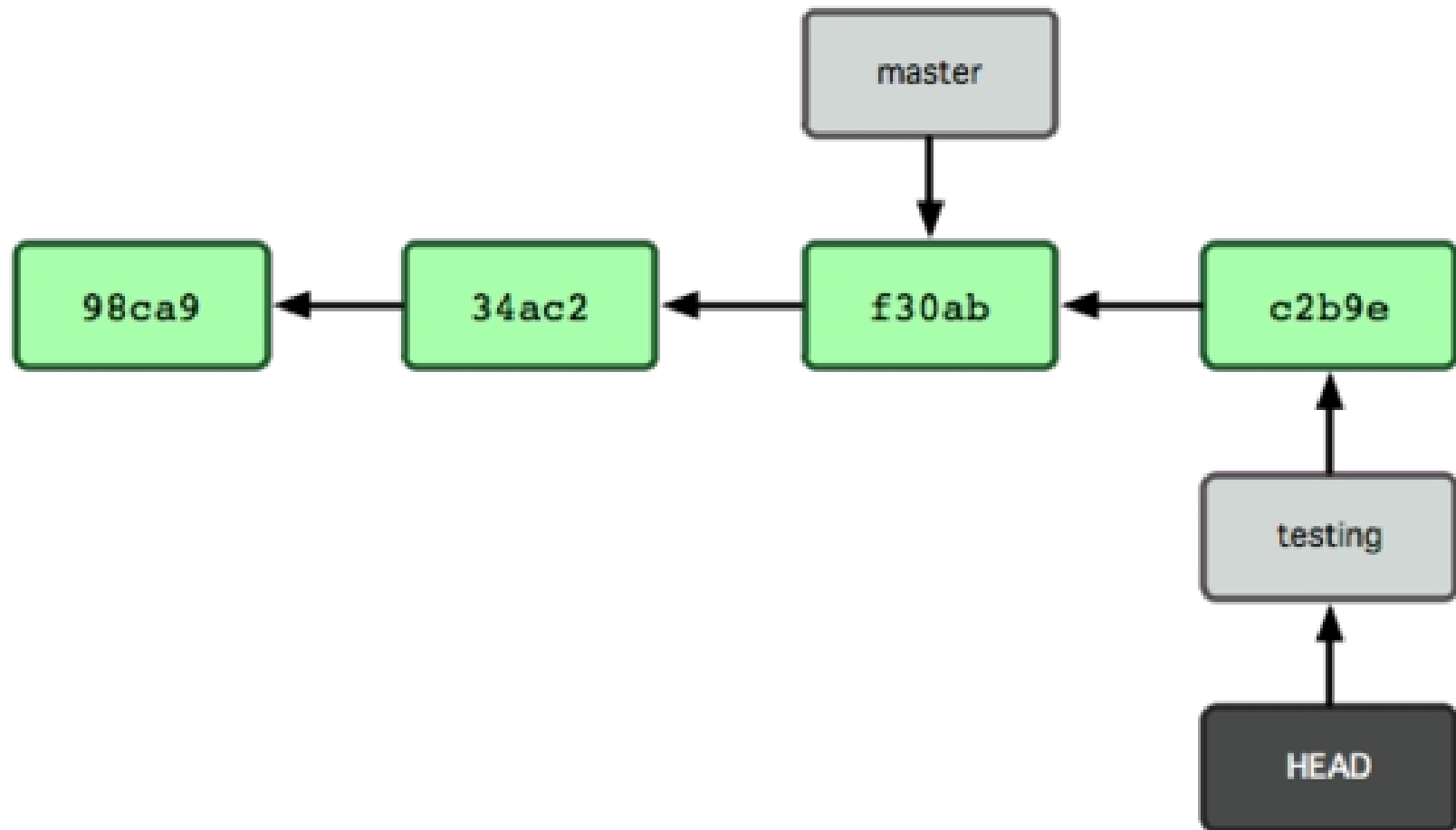
```
$ git branch testing
```

Branches are pointers



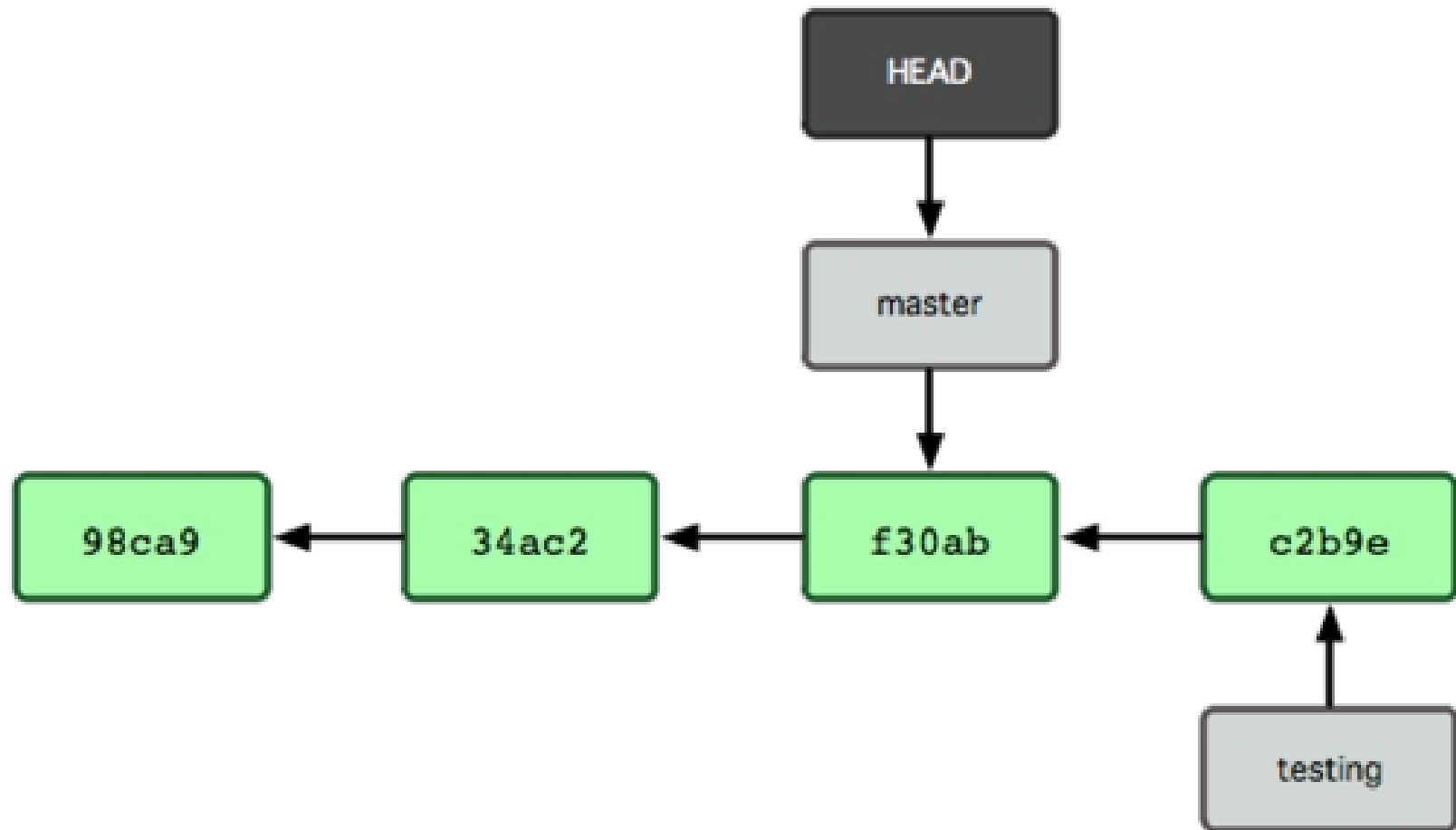
```
$ git checkout testing
```


Branches are pointers



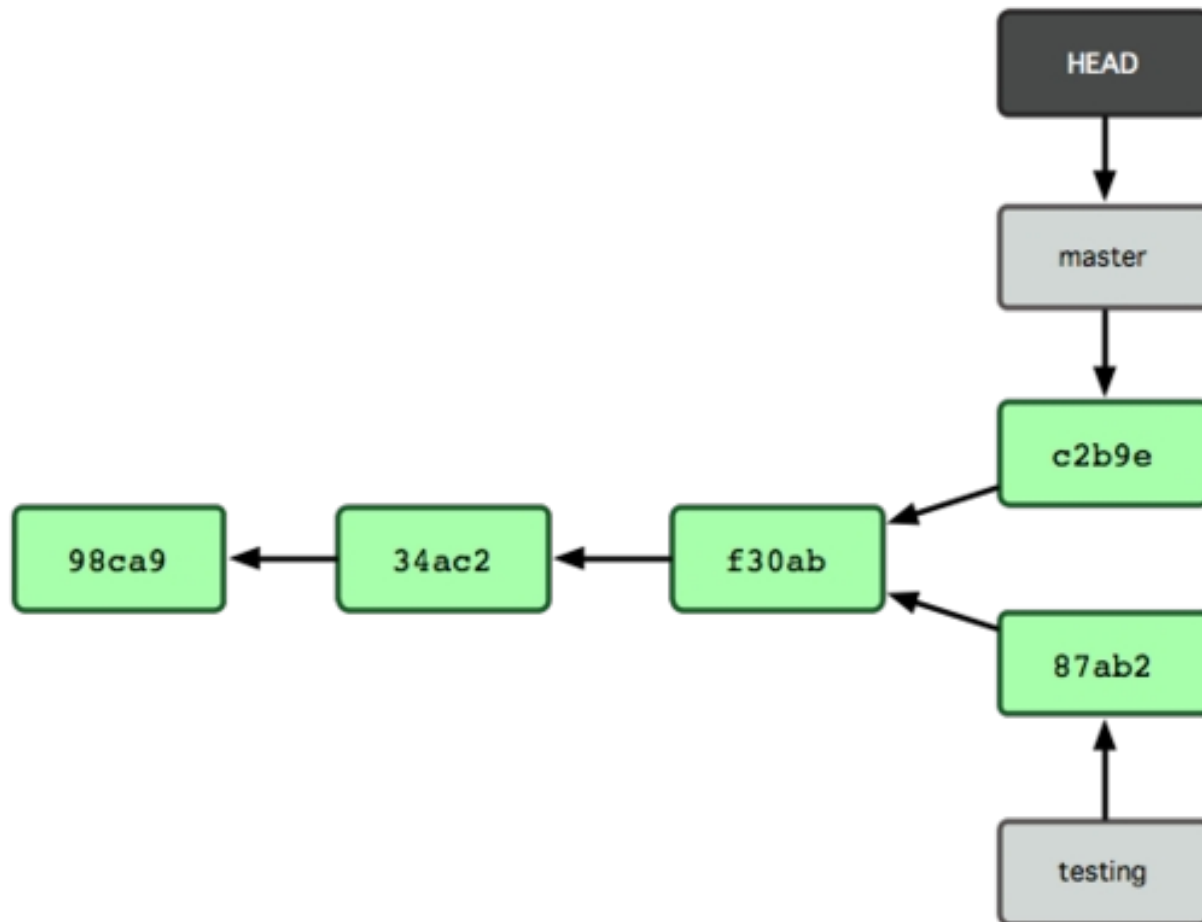
A new commit on the testing branch.

Branches are pointers



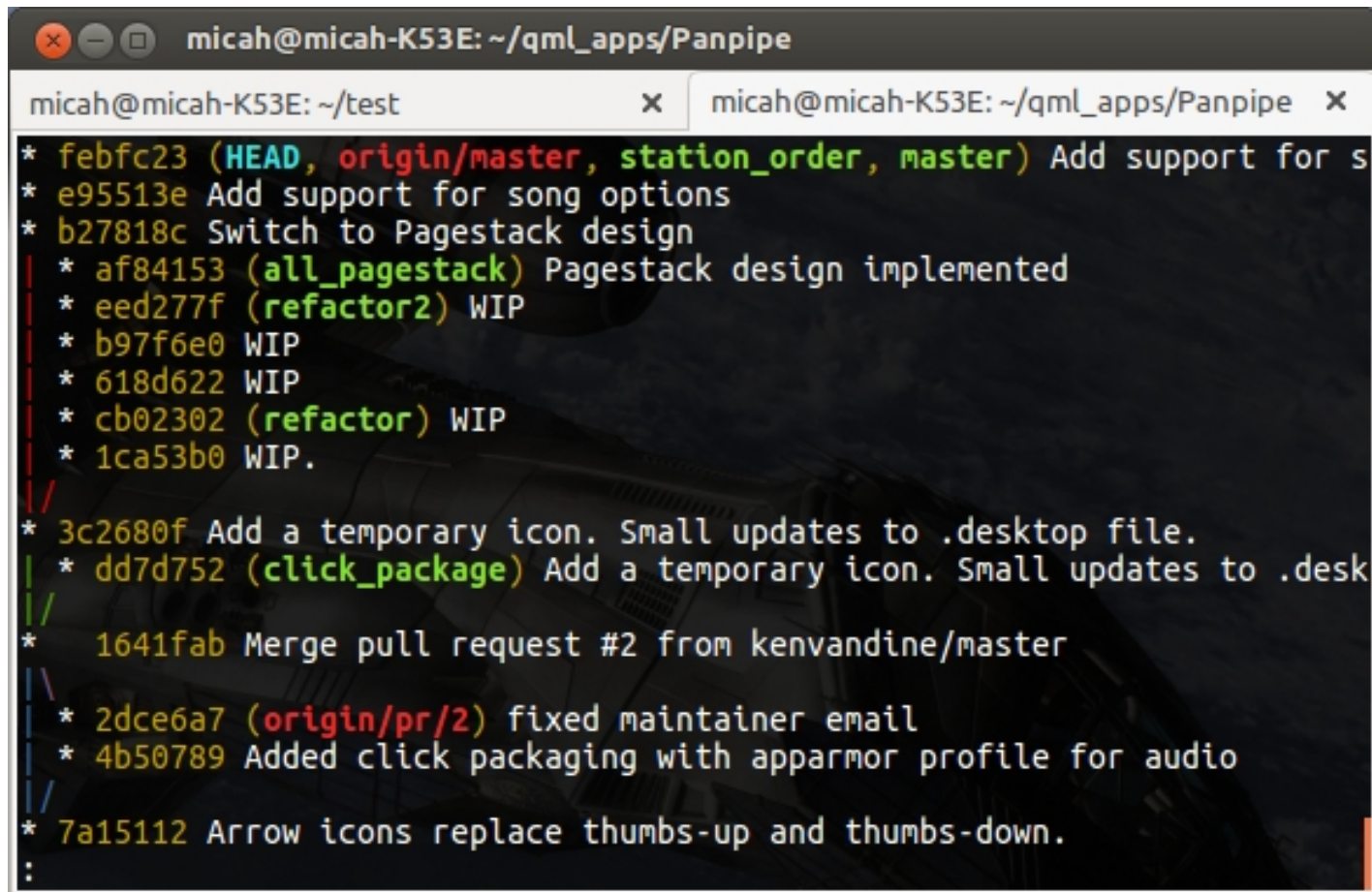
```
$ git checkout master
```

Branches are pointers



A new commit on the master branch.

Git Log EXTREME



```
micah@micah-K53E: ~/qml_apps/Panpipe
micah@micah-K53E: ~/test
micah@micah-K53E: ~/qml_apps/Panpipe
* febf23 (HEAD, origin/master, station_order, master) Add support for s
* e95513e Add support for song options
* b27818c Switch to Pagestack design
  * af84153 (all_pagestack) Pagestack design implemented
  * eed277f (refactor2) WIP
  * b97f6e0 WIP
  * 618d622 WIP
  * cb02302 (refactor) WIP
  * 1ca53b0 WIP.
* 3c2680f Add a temporary icon. Small updates to .desktop file.
  * dd7d752 (click_package) Add a temporary icon. Small updates to .desk
  * 1641fab Merge pull request #2 from kenvandine/master
  * 2dce6a7 (origin/pr/2) fixed maintainer email
  * 4b50789 Added click packaging with apparmor profile for audio
* 7a15112 Arrow icons replace thumbs-up and thumbs-down.
:
```

The git log command has some tricks up its' sleeves.

Regular and Canadian LOLs

...just copy the following into ~/.gitconfig for your full color git lola action:

[alias]

lol = log --graph --decorate --pretty=oneline --abbrev-commit

lola = log --graph --decorate --pretty=oneline --abbrev-commit --all

[color]

branch = auto

diff = auto

interactive = auto

status = auto

From Conrad Parker's blog <<http://blog.kfish.org/2010/04/git-lola.html>>

Git Diff

Git can show you the difference between two commits or two branches.

```
$ git diff 3df73768 f3c3bfc
```

- Show the differences between the two commits

```
$ git diff master testing
```

- Show the differences between the two branches.
Remember, branches are pointers to commits.
- The `--stat` flag shows a list of changed files rather than all the changes.

Lightweight Tags

A tag is a simple static reference to a commit. Think of it like a branch that doesn't move.

```
$ git tag
```

- List tags in the project

```
$ git tag v1.00
```

- Place a tag named “v1.00” on the current commit

In Class Exercise Part 2



Git Merge

The `git merge` command merges in the named branch into the branch that you are currently on.

```
$ git merge <other_branch_name>
```

- Creates a new commit with the two branches combined.

```
$ git merge --squash <branch_name>
```

- Doesn't make a new commit with two ancestors. Instead all the merge differences are added to the index, and you can commit them when ready.

In Class Exercise Part 3



Remote Repositories

“Remote repositories are versions of your project that are hosted on the Internet or network somewhere.”

– Scott Chacon, Pro Git

Git Remote

```
$ git remote -v
```

- List all remote servers (-v means show remote url)

```
$ git remote add <local_name> <url>
```

- Add the repo at the given <url> as a remote server and call it by <local_name> in this repo.

Git Fetch

The git fetch command updates your repository's information about the remote repository. It will be easiest to observe the remote branches with the 'git lola' command.

```
$ git remote add <remote_name> <url>
```

```
$ git fetch <remote_name>
```

Git Push

The git push command puts/updates your local branch to a branch on the remote repository.

```
$ git push <remote_name> <local_branch_name>
```

- Add/update your <local_branch_name> on the remote server.

```
$ git push <remote_name>
```

```
<local_branch_name>:<remote_branch_name>
```

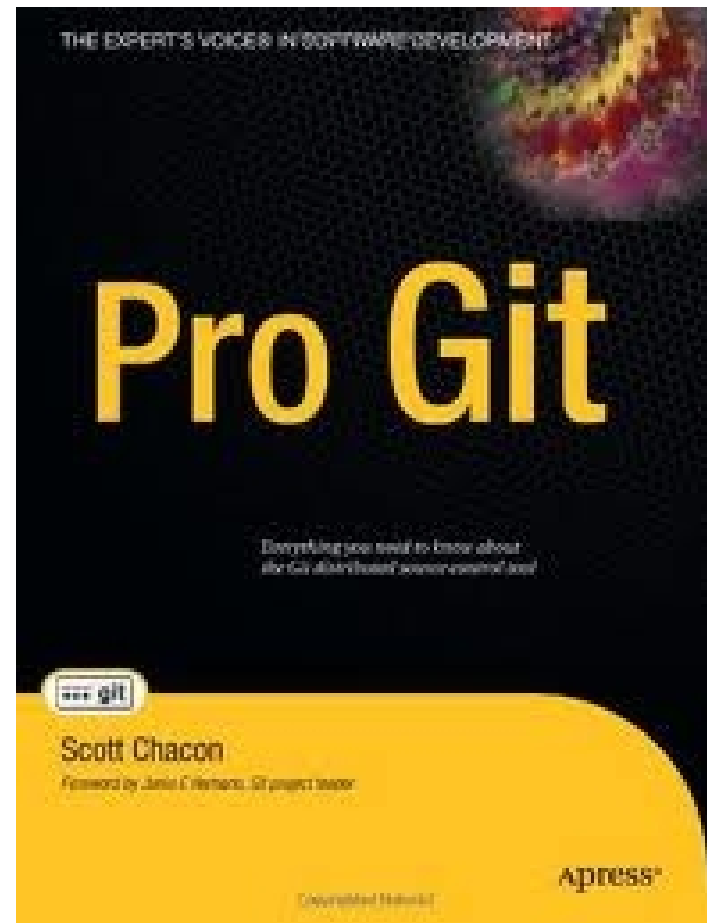
- Same as above, but rename branch on remote

In Class Exercise Part 4



What You Need Now

- More practice
- Pro Git by Scott Chacon
 - \$23 on Amazon
 - PDFs online for free



Git Questions?

Git Out Of Here

- I can linger for a while if you have further questions.
- Thank you for having me, it's a pleasure and honor to be here.
- Go into the world and use version control! It takes very little effort and does so much for you!