

Distributed version control with Git



Part 1:
Working locally

Daniel Sussman

<https://gitlab.com/dmsussman/workingWithGit/>

Common commands / outline of talk

Command	Description
git init	Initialize a git repository / this talk
git add <i>files</i>	Add the files to the staging area
git status	View the status of files in working directory and staging area
git commit -m " <i>message</i> "	Record a snapshot of the staging area
git checkout	Switch branches or checkout a file / commit
git log, git diff	View a record of commits; see what is different between versions of a file
git help [<i>verb</i>]	Get help on the specified command

This is just scratching the surface!
More git resources:

Git's own documentation!

Just type “git help” or “git help [command]”

The “Pro Git” book

<https://git-scm.com/book/en/v2>

TryGit for an interactive walkthrough of basic commands

<https://try.github.io/>

Atlassian has some nice tutorials for learning git

<https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud>

In addition to the above references, I drew inspiration and/or stole slide ideas from:

Zachary Ling, “Fundamentals of Git”

<http://slideplayer.com/slide/6171599/>

Ruth Anderson’s CSE 390a lecture:

<https://courses.cs.washington.edu/courses/cse390a/>

https://github.com/ldfaiztt/CSE390A/blob/master/Week%201/390aGitIntro_12au.pdf



What is version control?

A (systematic) way of managing multiple versions of programs, documents, databases, etc.

Theorem: Almost all real projects use some kind of version control.

Corollary:

Why use version control?

Team work:

- * Makes working collaboratively much easier

Individual work:

- * Scientific reproducibility of code
- * Magical time machine for reverting to previous version of code



Git is a distributed system for version control

Distributed:

- No copy of a repository is more important than others (except by convention)

- Get version control even if offline

Git advantages:

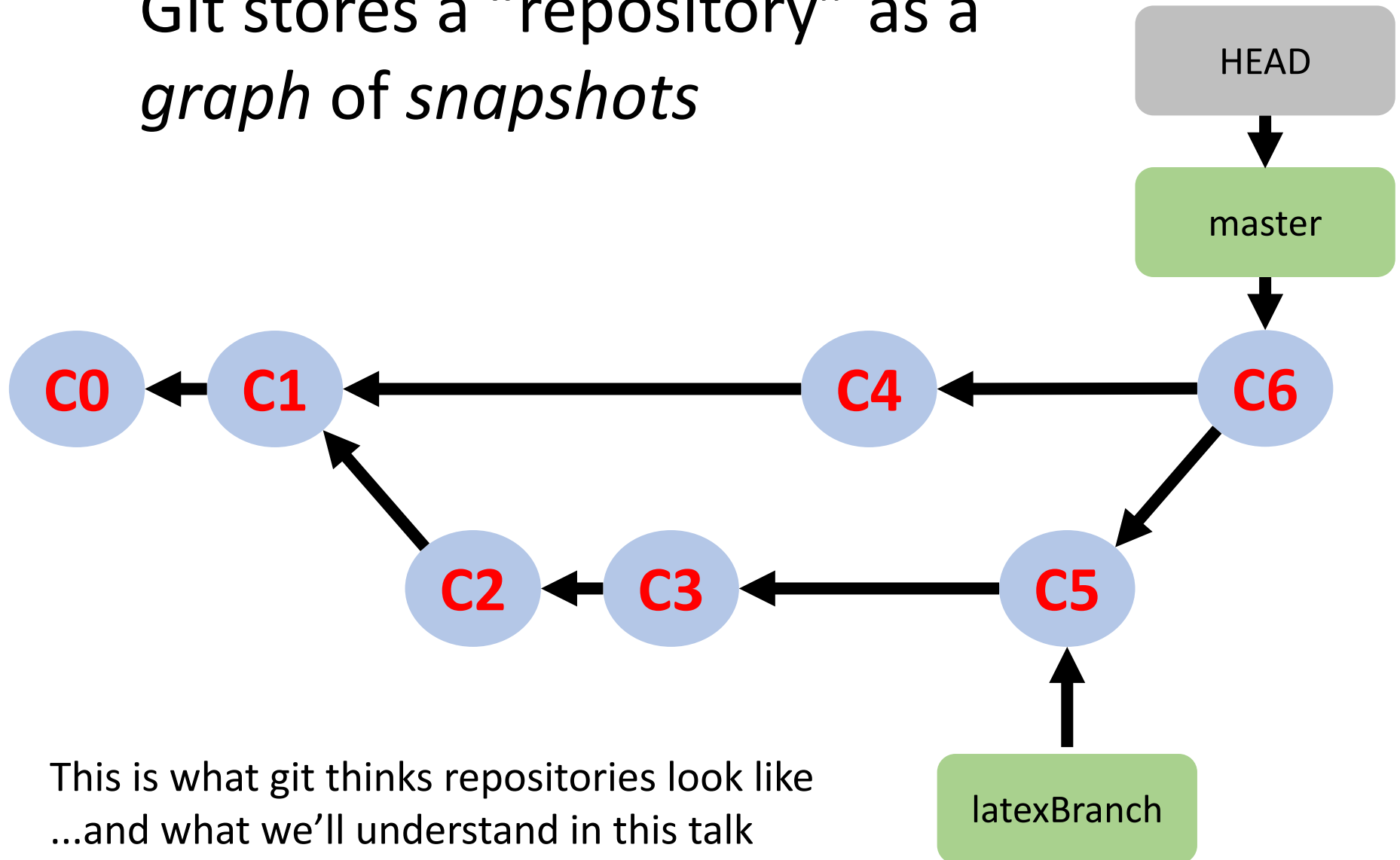
- Very resilient (because it's distributed)

- Very fast

- Very compressed (favors frequent, small updates)

- Very widely used (every git question, ever, has been asked on Stack Overflow)

Git stores a “repository” as a
graph of snapshots



This is what git thinks repositories look like
...and what we'll understand in this talk

Step 0: install git

Is git already on your computer?

Type "git --version" on the command line to find out

If not...

Windows: <https://git-for-windows.github.io/>

Mac: <https://sourceforge.net/projects/git-osx-installer/files/>

or "brew install git" or...

Linux: "sudo apt-get install git" or "sudo yum git" or ...

Set default user name and email

```
$ git config --global user.name "My Name"
```

```
$ git config --global user.email "myEmail@email.email"
```

Starting a new repository: the plan

Our basic workflow will be:

(start a repository)

Edit some files

Stage those changes

Review those changes

Commit those changes

... and repeat

Starting a new repository: *git init*

Let's navigate to a blank directory and begin...

```
$ git init
```

We've just initialized an empty repository!

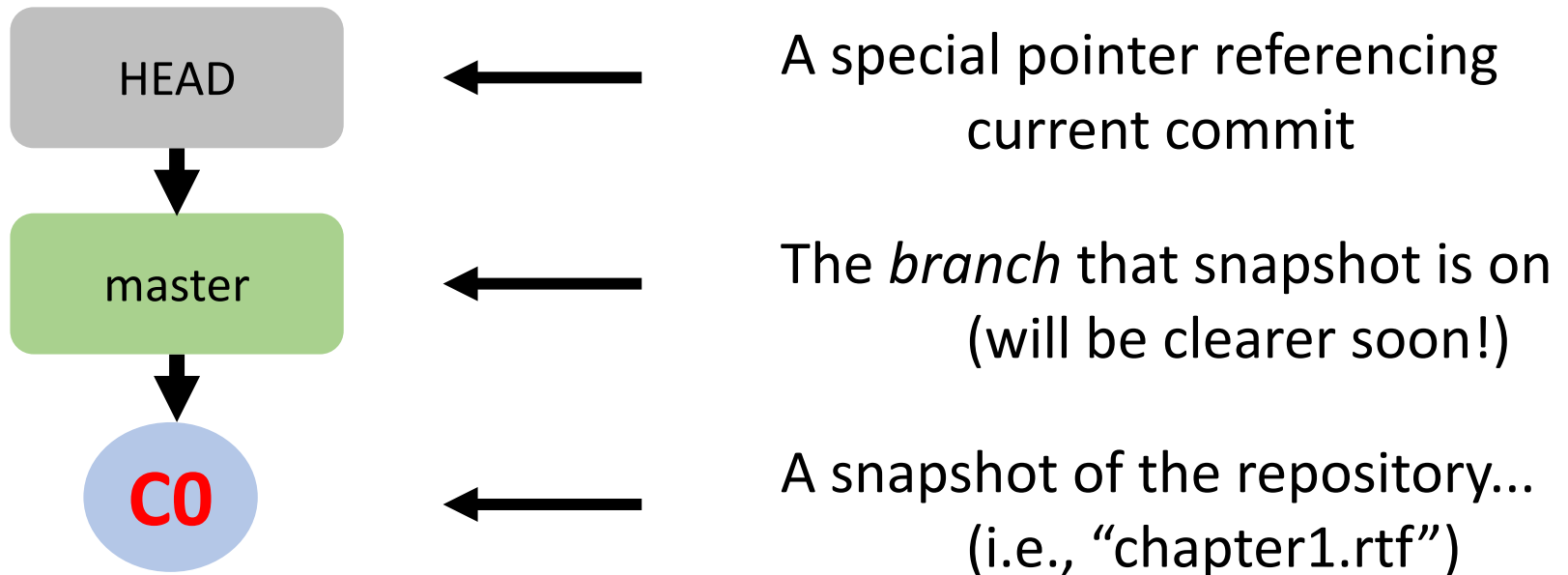
Starting a new repository: *git init*

*Hey, it turns out I'm writing my Ph.D. thesis... in vim
...let's start with a few mystical commands*

```
$ vim chapter1.rtf  
$ git add chapter1.rtf  
$ git commit -m "Chapter 1 finished"
```

*Version control does not just have to be code!
We'll follow this pretend project for the rest of the talk*

Git's thoughts about this repository



The (local) trees of git

The “HEAD”

The last committed snapshot of the repository

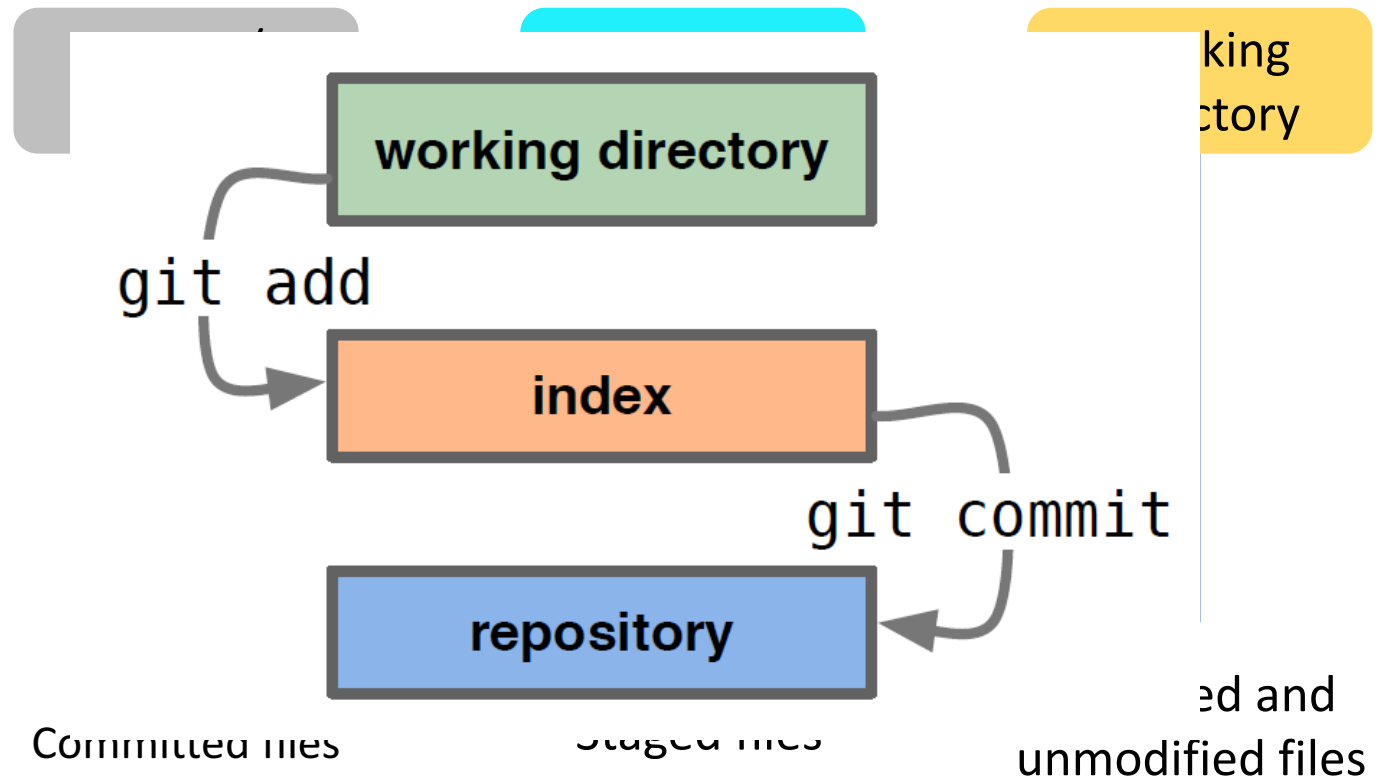
The “Staging area” (or “Index”)

A proposed next snapshot

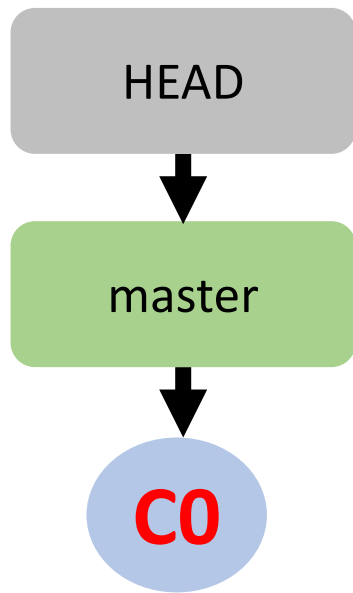
The “Working directory” (or “working tree”)

Where you do work!

The (local) trees of git



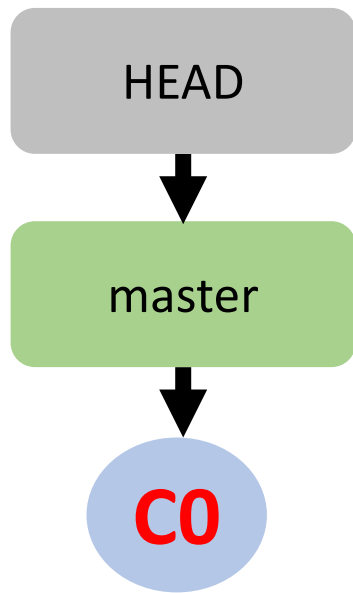
Looking at the current status: *git status*



```
$ git status
On branch master
nothing to commit, working tree clean
```



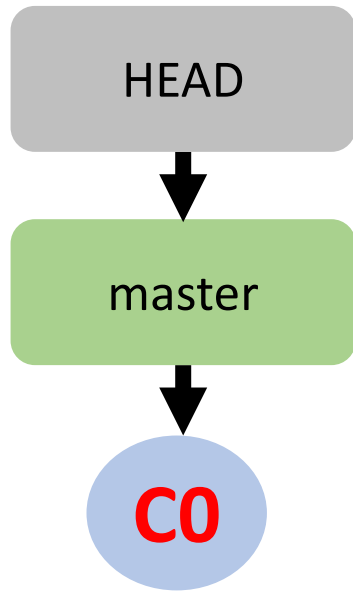
Life cycle of a commit: Editing files



```
$ vim chapter2.rtf
```



Life cycle of a commit: seeing changes



\$ git status

On branch master

Untracked files: (use "git add <file>..." to include in what will be committed)

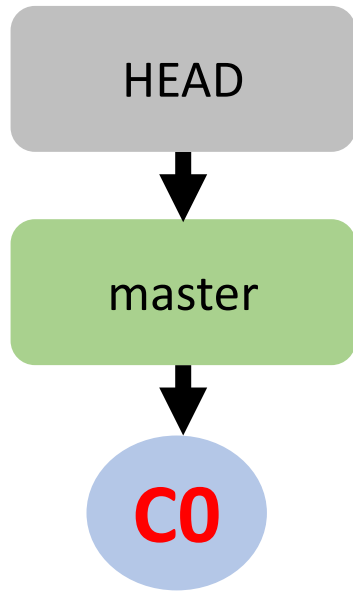
chapter2.rtf

nothing added to commit but untracked files present (use "git add" to track)

Look at all those helpful messages!



Life cycle of a commit: *git add*



git add stages files
to be committed

```
$ git add chapter2.rtf
```

```
$ git status
```

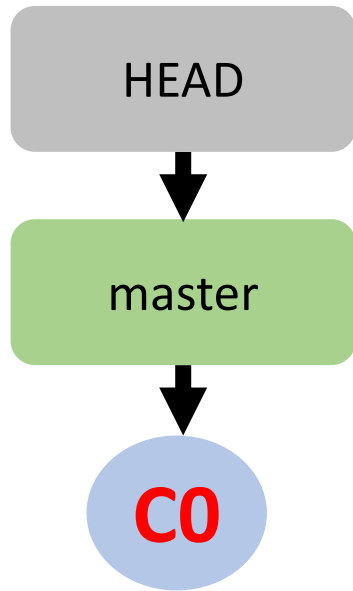
On branch master

Changes to be committed: (use "git reset
HEAD <file>..." to unstage)

new file: chapter2.rtf



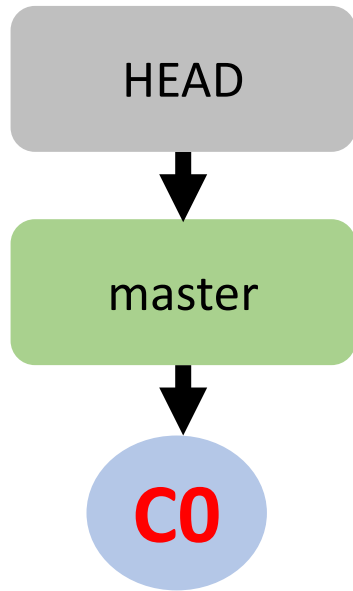
Life cycle of a commit: *git status*



```
$ vim chapter3.rtf
$ git status
On branch master
Changes to be committed:
  new file:   chapter2.rtf
Untracked files:
  chapter3.rtf
```



Life cycle of a commit: *git status*



```
$ git add chapter3.rtf
```

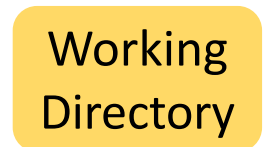
```
$ git status
```

On branch master

Changes to be committed:

new file: chapter2.rtf

new file: chapter3.rtf

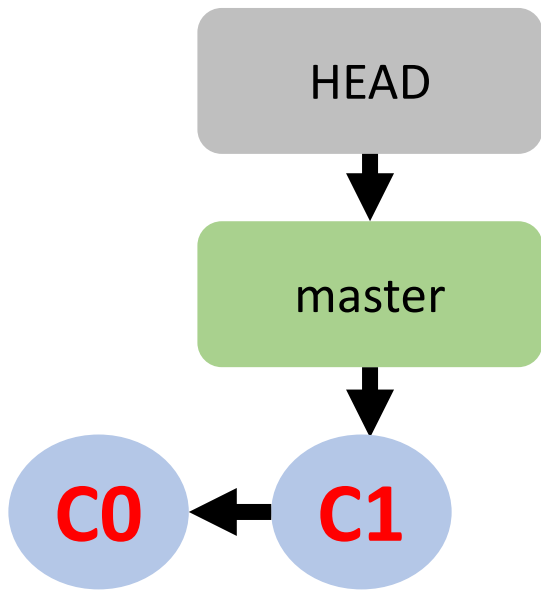


chapter1.rtf

chapter2.rtf

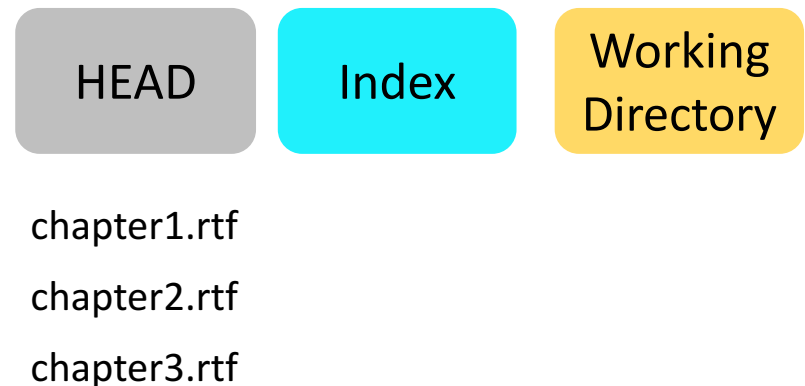
chapter3.rtf

Looking at the current status: git status



```
$ git commit -m "Chapters 2 and 3 written"
$ git status
On branch master
nothing to commit, working tree clean
```

git commit takes a new snapshot
...commit messages are **mandatory**
(and useful!)



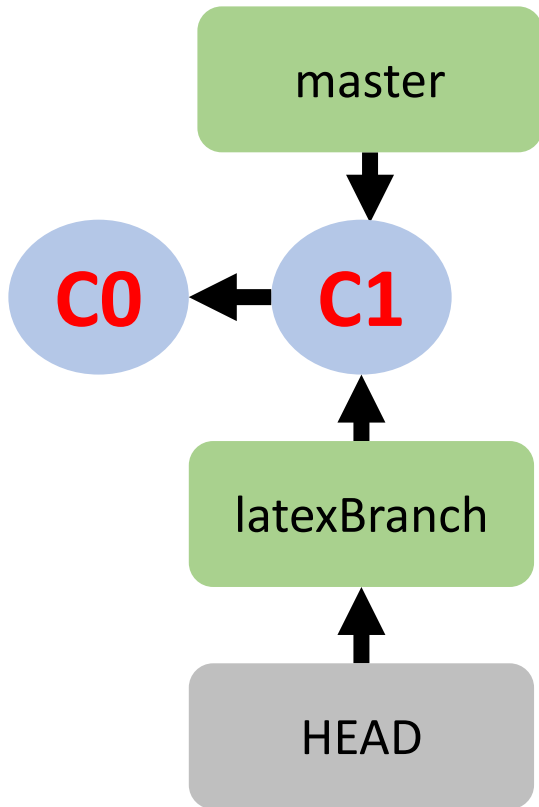
Branches!

After much thought, I've decided ".rtf" format is terrible for a thesis...

...if only I could experiment with an alternative



Making a new branch: *git checkout*



```
$ git checkout -b latexBranch
```

Switched to a new branch 'latexBranch'

The “-b” option tells git to make a new branch... we'll see that checkout is quite powerful

HEAD

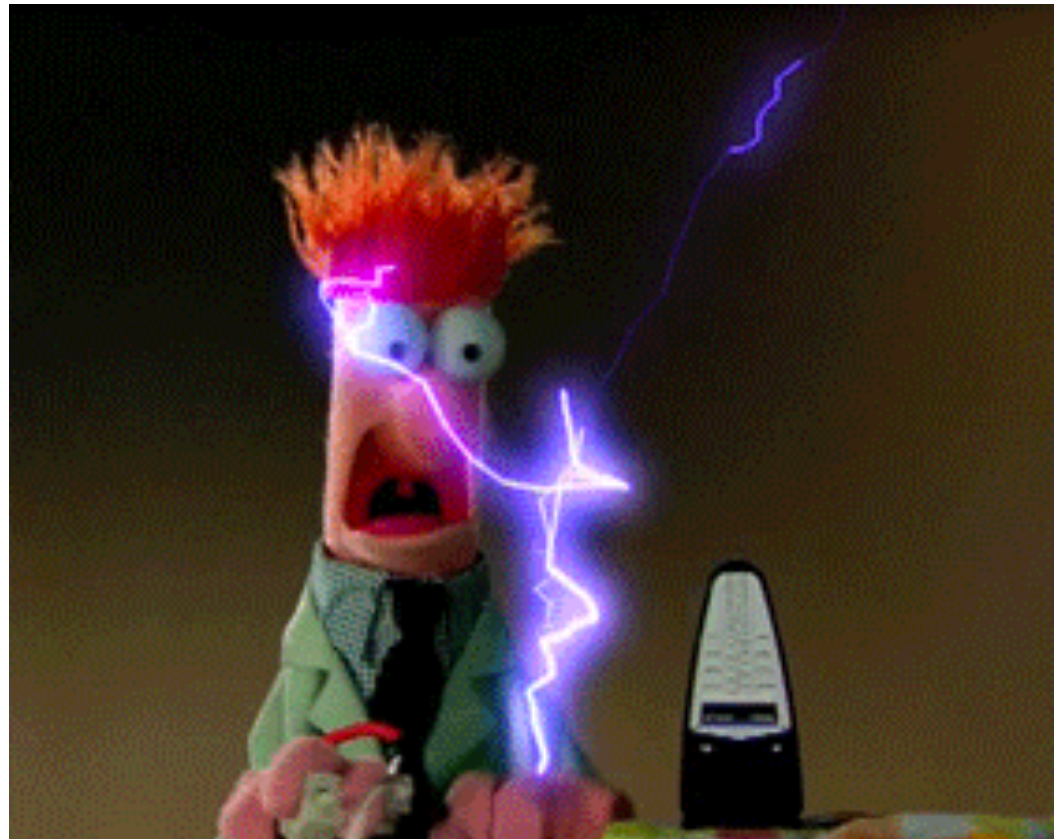
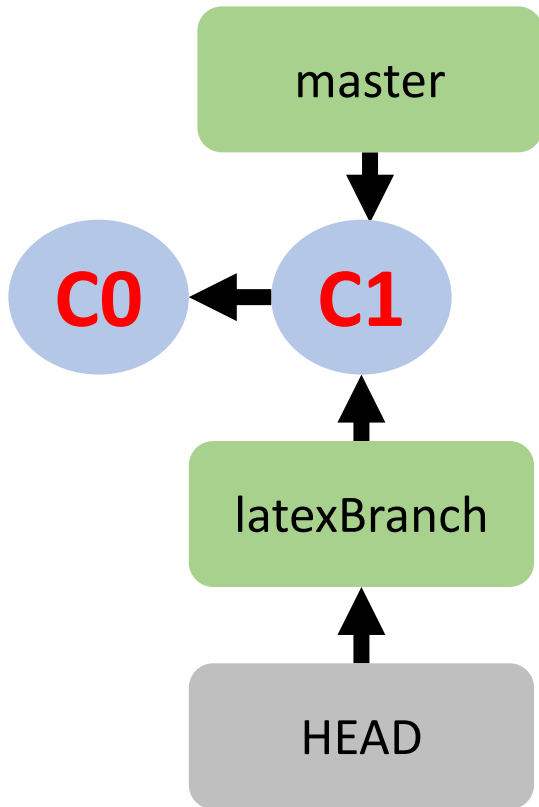
Index

Working
Directory

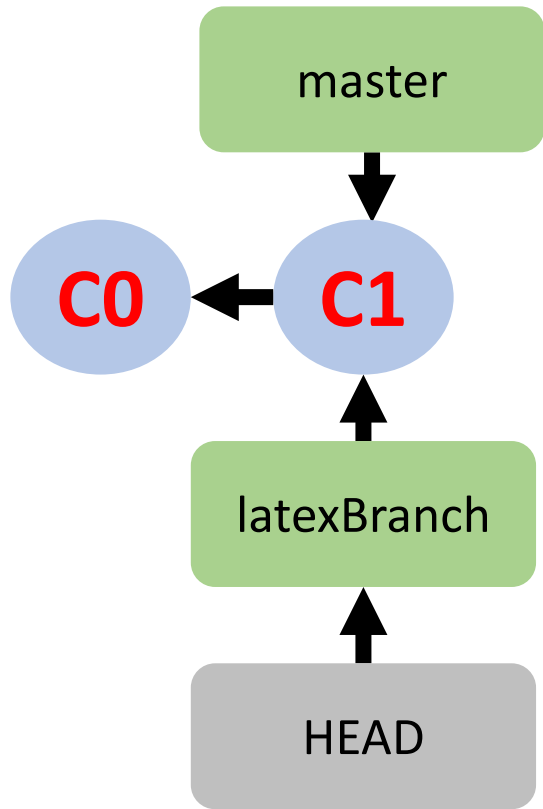
chapter1.rtf
chapter2.rtf
chapter3.rtf

Rescuing me from myself: *git checkout*

```
$ rm chapter1.rtf
```



Rescuing me from myself: *git checkout*



\$ git status

On branch latexBranch

Changes not staged for commit: (use "git checkout -- <file>..." to discard changes in working directory)

deleted: chapter1.rtf

no changes added to commit

HEAD

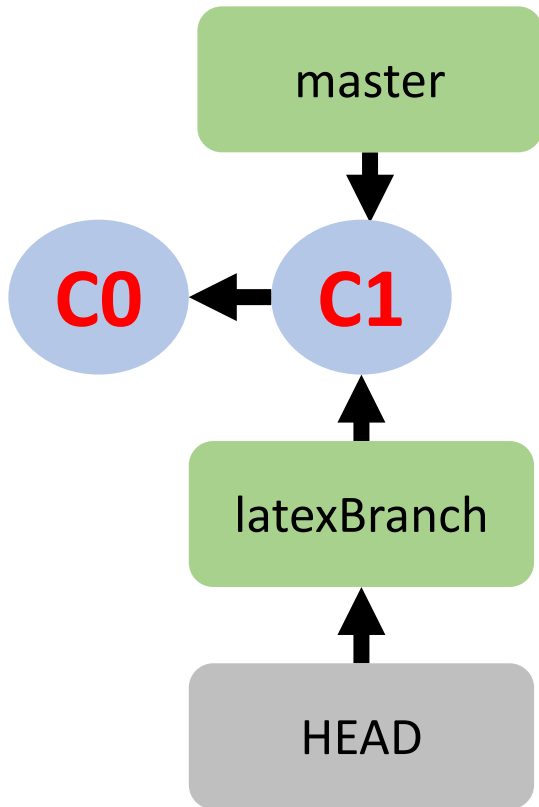
Index

Working
Directory

chapter1.rtf
chapter2.rtf
chapter3.rtf

chapter1.rtf

Rescuing me from myself: *git checkout*



```
$ git checkout -- chapter1.rtf  
$ mv chapter1.rtf chapter1.tex  
$ vim chapter1.tex
```

HEAD

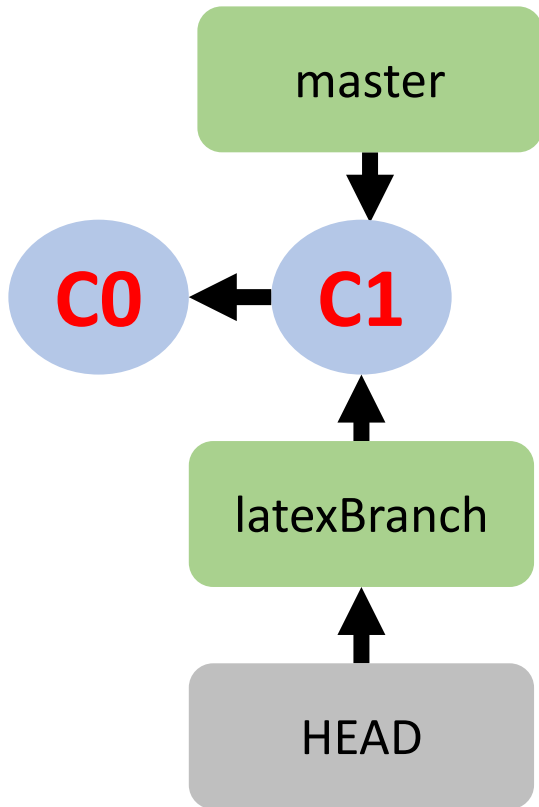
Index

Working
Directory

chapter1.rtf
chapter2.rtf
chapter3.rtf

chapter1.tex

Preparing for a new commit...



\$ git status

On branch latexBranch

Changes not staged for commit:

deleted: chapter1.rtf

Untracked files:

chapter1.tex

HEAD

Index

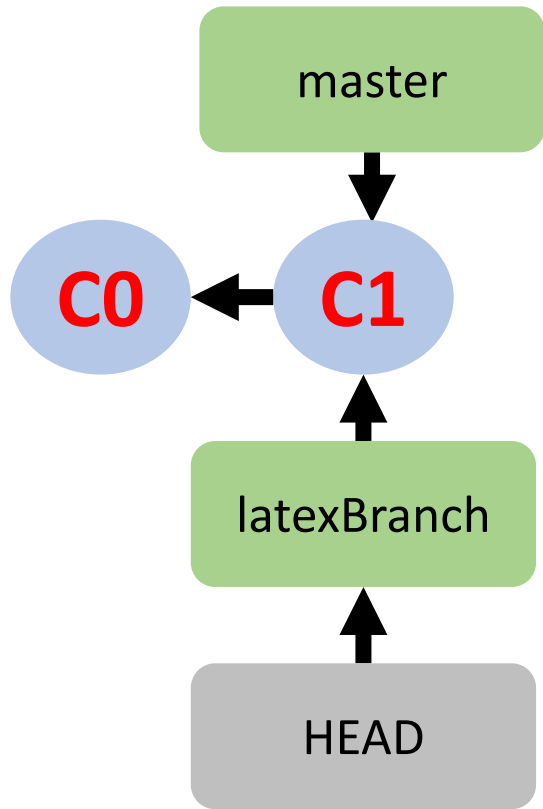
Working
Directory

chapter1.rtf
chapter2.rtf
chapter3.rtf

chapter1.tex

Preparing for a new commit...

```
$ git add --all
```



HEAD

Index

Working
Directory

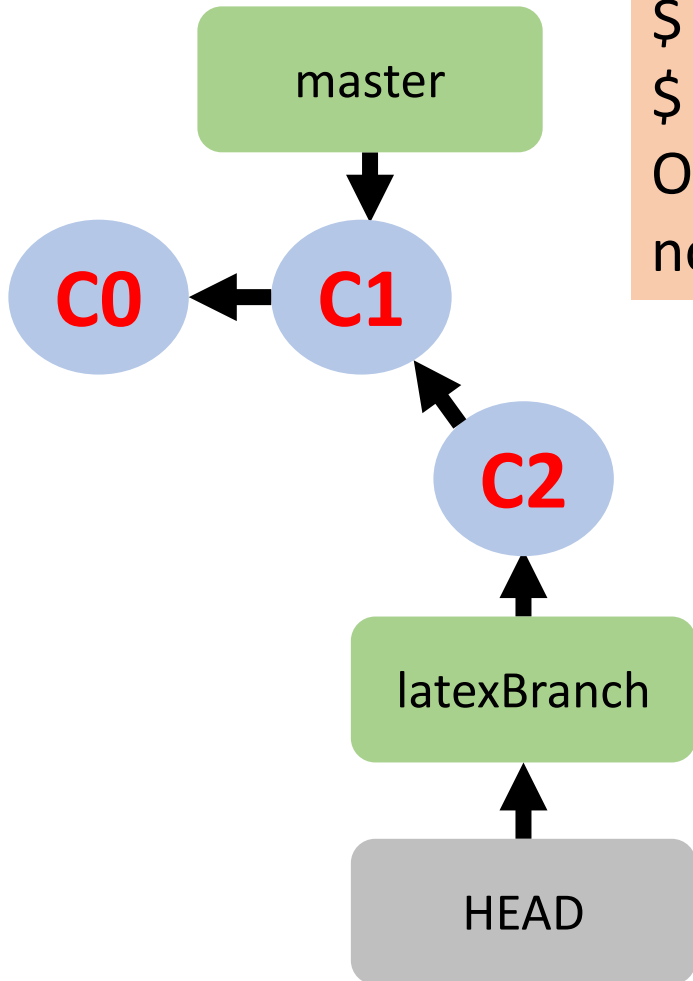
chapter1.rtf

chapter1.tex

chapter2.rtf

chapter3.rtf

Committing the new branch



```
$ git commit -m "re-wrote chapter 1 in tex"
$ git status
On branch latexBranch
nothing to commit, working tree clean
```

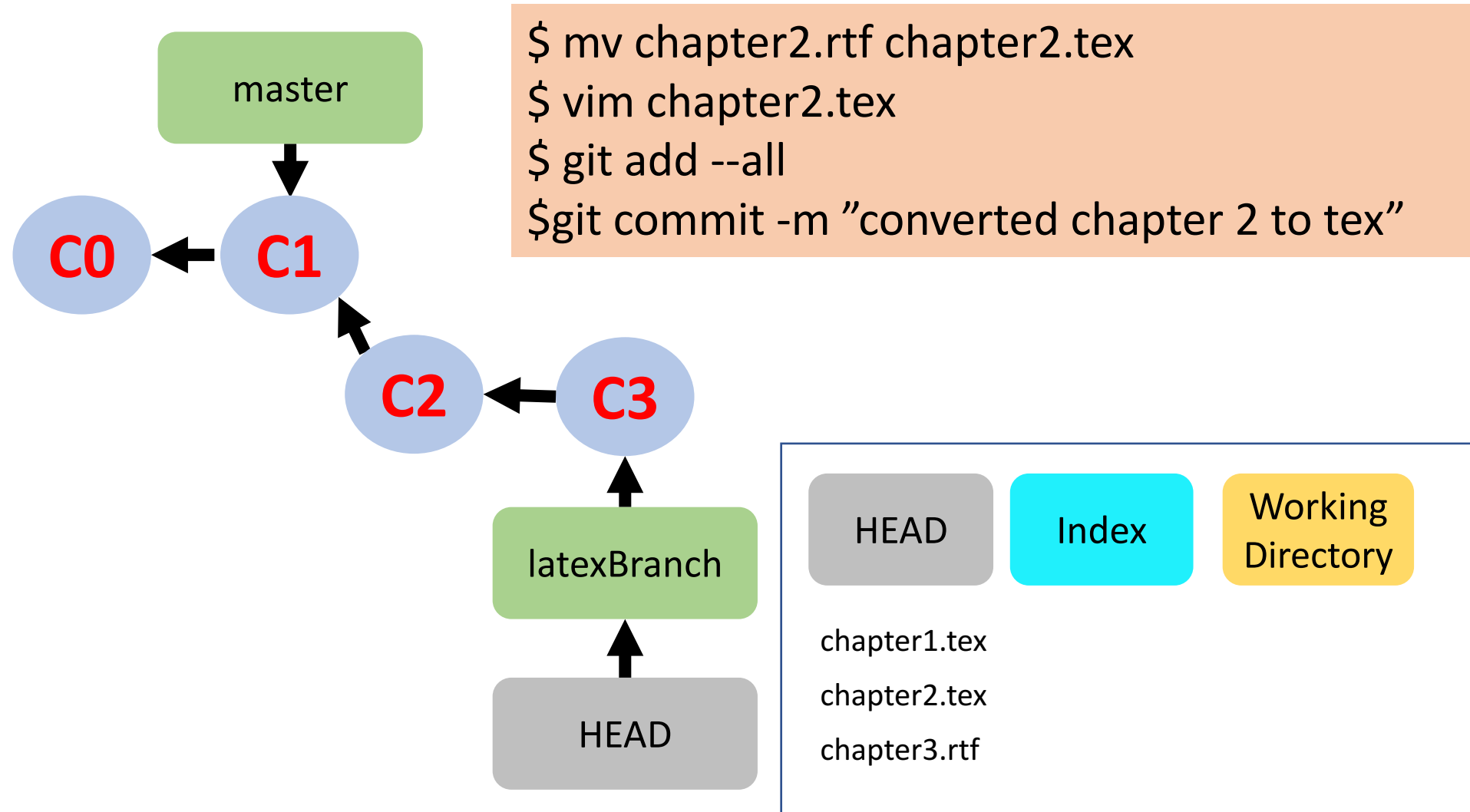
HEAD

Index

Working
Directory

chapter1.tex
chapter2.rtf
chapter3.rtf

Extending the branch



Moving around the git tree: *git checkout*

*My advisor just asked for **another** chapter, and I haven't finished converting everything to TeX yet!*



Moving around the git tree: *git checkout*

HEAD

```
graph TD; HEAD[HEAD] --> master[master]; master --> C1((C1)); C1 --> C0((C0)); C2((C2)) --> C1; C3((C3)) --> C2; latexBranch[latexBranch] --> C3;
```

\$ git checkout master
Switched to branch 'master'

All of our old .rtf files are still there,
in exactly the state we left them in!

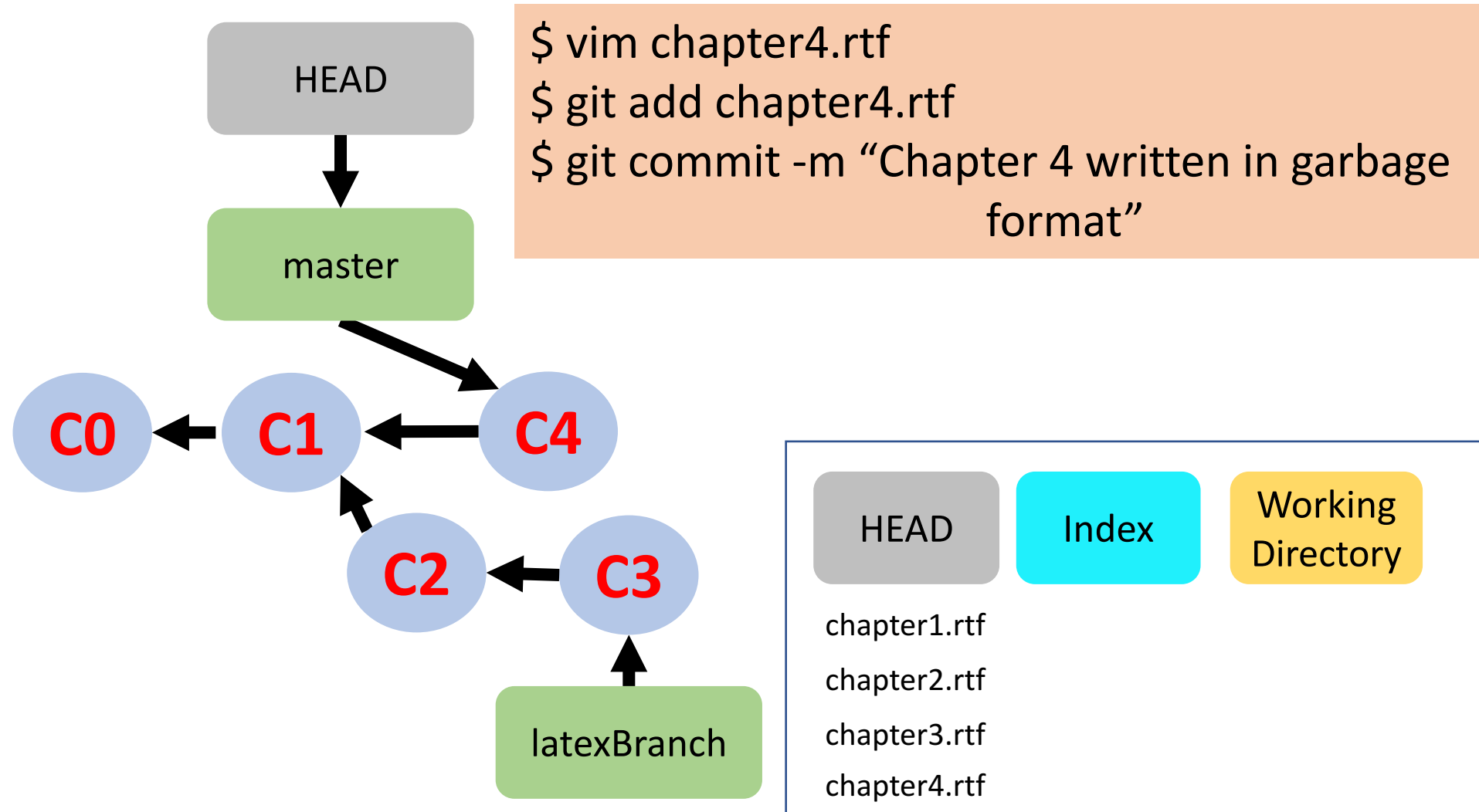
HEAD

Index

Working
Directory

chapter1.rtf
chapter2.rtf
chapter3.rtf

Moving around the git tree: *git checkout*



“Where am I?": *git branch*

HEAD

```
$ git branch  
  latexBranch  
* master
```

master

I guess we're on the “master” branch

C0

C1

C4

C2

C3

latexBranch

HEAD

Index

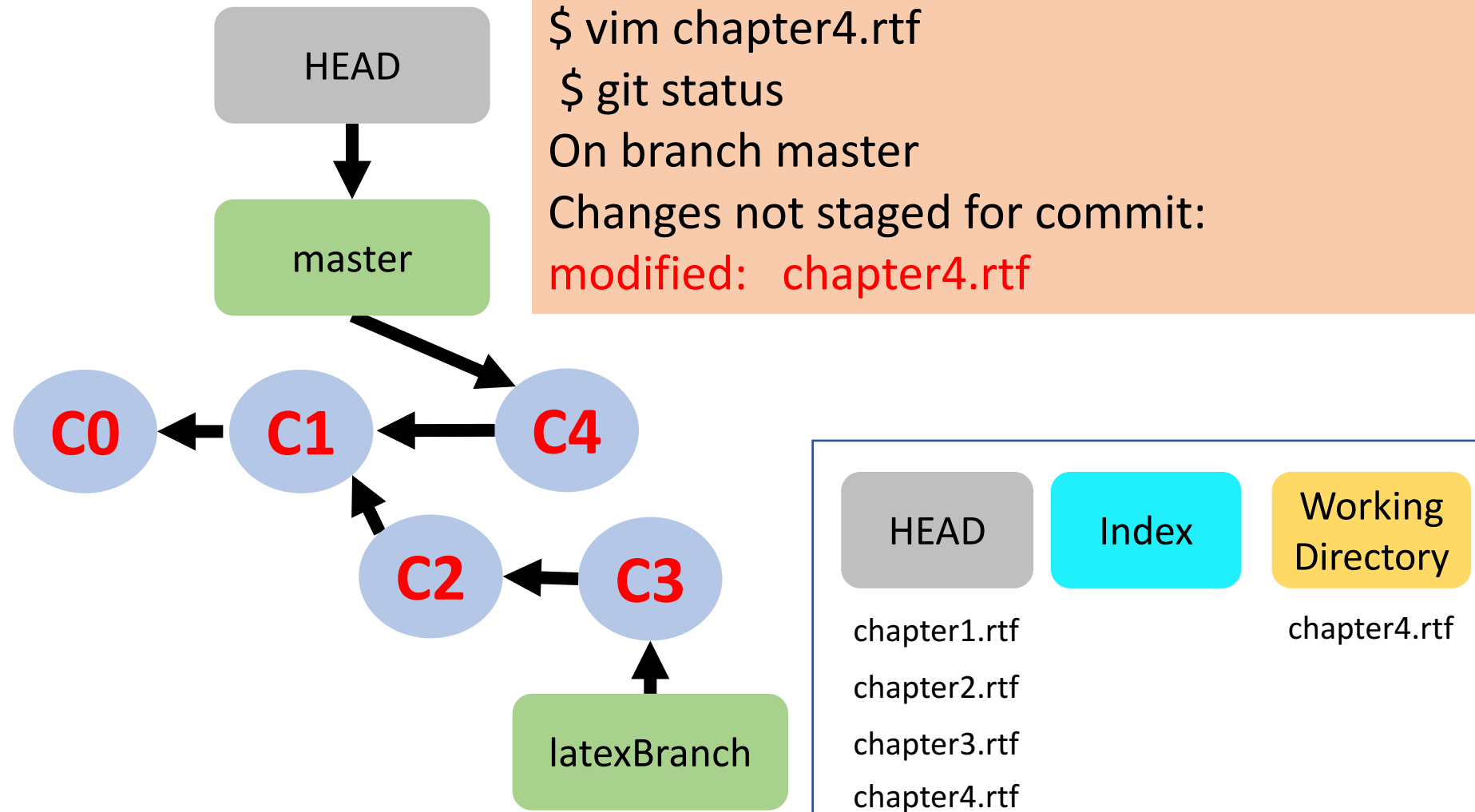
Working
Directory

chapter1.rtf
chapter2.rtf
chapter3.rtf
chapter4.rtf

“What have I done?": *git diff*

Let's edit one of our files

```
$ vim chapter4.rtf
$ git status
On branch master
Changes not staged for commit:
  modified:   chapter4.rtf
```



“What have I done?": *git diff*

Let's edit one of our files

```
$ git diff
```

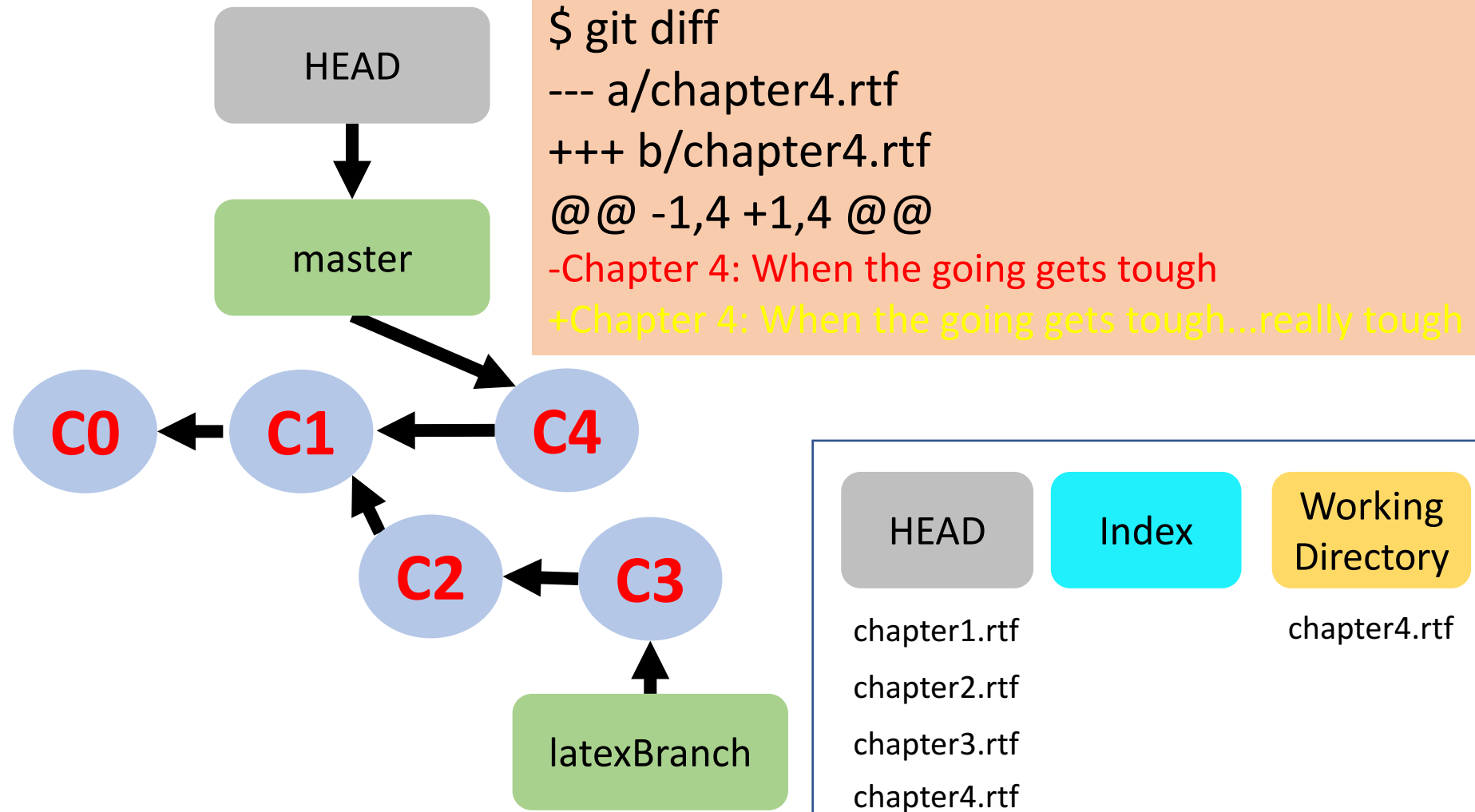
```
--- a/chapter4.rtf
```

```
+++ b/chapter4.rtf
```

```
@@ -1,4 +1,4 @@
```

```
-Chapter 4: When the going gets tough
```

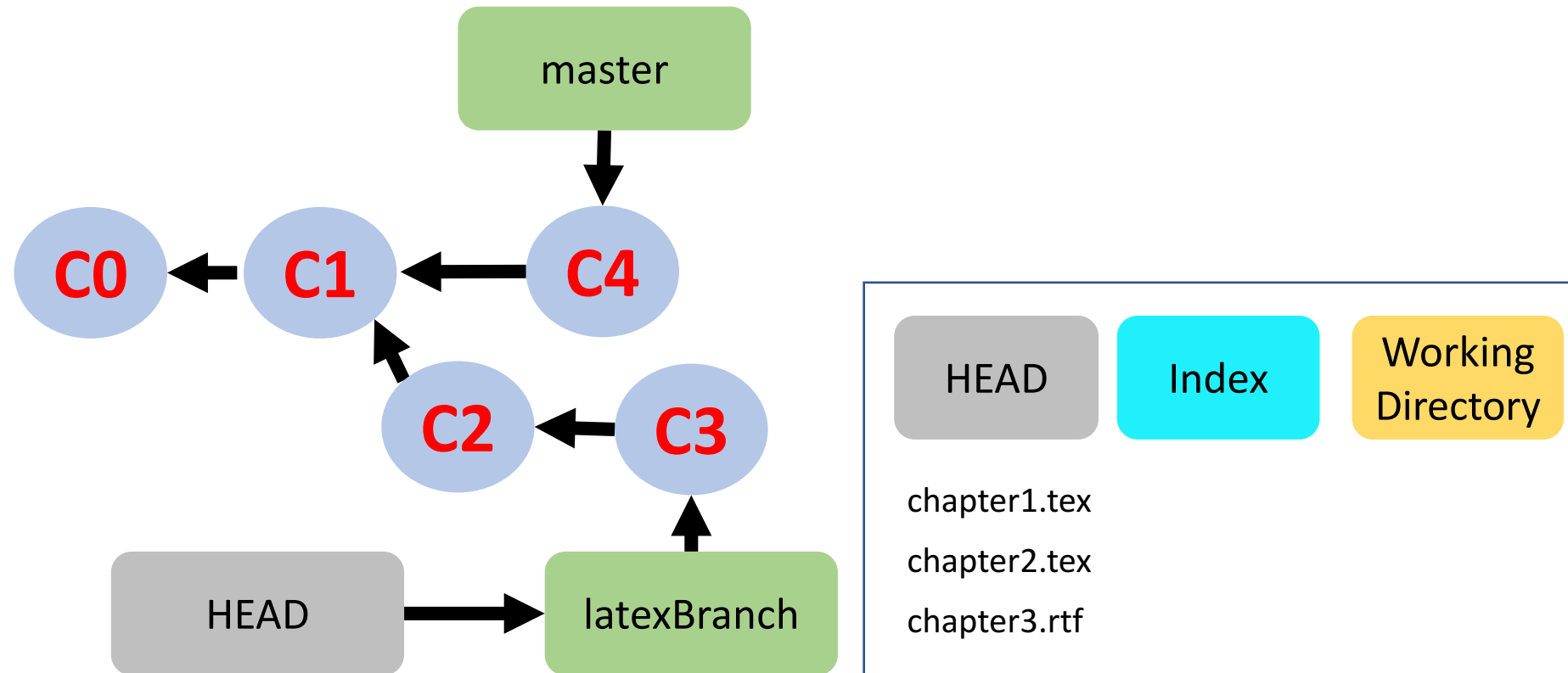
```
+Chapter 4: When the going gets tough...really tough
```



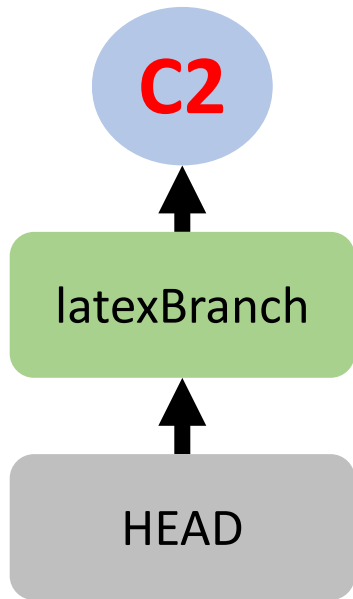
Undoing changes: *git checkout*

That edit wasn't so edifying.... let's undo it and switch back to the latexBranch

```
$ git checkout -- chapter4.rtf  
$ git checkout latexBranch
```



“What have I done?": *git log*



This would put you in a “detached head” state...

```
$ git log
```

```
commit 2b72641b21f9873470d2b2493e4030f92fb0625d (HEAD -> latexBranch)
Author: Daniel Sussman <dmsussma@syr.edu>
Date: Tue Sep 12 12:20:49 2017 -0400
    converted chapter 2 to tex

commit e69f620b9064b3c58e0b5df7fae80b956aa6ef4d
Author: Daniel Sussman <dmsussma@syr.edu>
Date: Tue Sep 12 12:15:36 2017 -0400
    re-wrote chapter 1 in tex

commit 0173951
Author: Daniel
Date: Tue Sep
    Chapters 2

commit 07c5f4403e2ef9f3ba34194c1ddc49955174786b
Author: Daniel Sussman <dmsussma@syr.edu>
Date: Tue Sep 12 10:39:24 2017 -0400
    Chapter 1 finished
```

Just like files and branches, specific commits can also be checked out by their *commit hash*:

```
$ git checkout e69f620
```

(typically just the first 7 characters needed)

“What have I done?”: *git log*

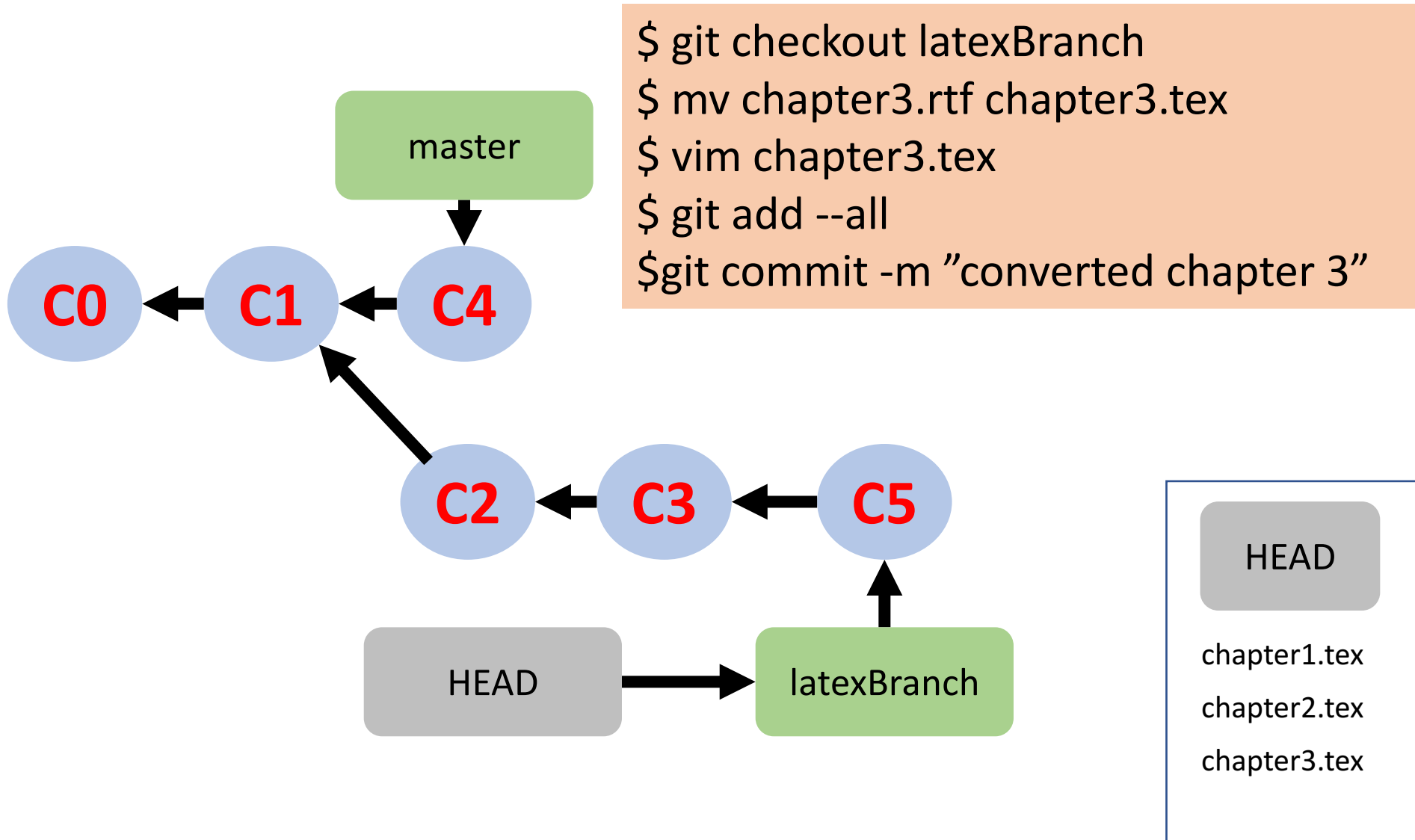
```
$ git lg2
```

```
lg2 = log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold cyan)%aD%C(reset)  
%C(bold green)(%ar)%C(reset)%C(bold yellow)%d%C(reset)%n' %C(white)%s%C(reset) %C(dim white)- %an%C(reset)' --all
```

```
* bcb0fdb - Tue, 12 Sep 2017 12:40:31 -0400 (9 minutes ago) (master)
|         Chapter 4 written in garbage format - Daniel Sussman
| * 2b72641 - Tue, 12 Sep 2017 12:20:49 -0400 (29 minutes ago) (HEAD -> latexBranch)
| |         converted chapter 2 to tex - Daniel Sussman
| * e69f620 - Tue, 12 Sep 2017 12:15:36 -0400 (34 minutes ago)
|/         re-wrote chapter 1 in tex - Daniel Sussman
* 0173951 - Tue, 12 Sep 2017 11:31:42 -0400 (78 minutes ago)
|         Chapters 2 and 3 written - Daniel Sussman
* 07c5f44 - Tue, 12 Sep 2017 10:39:24 -0400 (2 hours ago)
|         Chapter 1 finished - Daniel Sussman
```

...git is extraordinarily feature-rich, and can take a very long time to learn. The most essential features, though, are straightforward

Let's finish this thesis...



Bringing branches back together: *git merge*

*Okay, I think it's time to merge the experimental
LaTeX branch with the rest of my thesis!*



Bringing branches back together: *git merge*

```
$ git checkout master  
$ git merge latexBranch
```

```
[dmsussma@~/repos/introToGitManningGroup]$ git merge latexBranch  
Auto-merging chapter3.tex  
Auto-merging chapter2.tex  
Auto-merging chapter1.tex  
Merge made by the 'recursive' strategy.  
chapter1.rtf => chapter1.tex | 2 +  
chapter2.rtf => chapter2.tex | 2 +  
chapter3.rtf => chapter3.tex | 2 +  
3 files changed, 3 insertions(+), 3 deletions(-)  
rename chapter1.rtf => chapter1.tex (93%)  
rename chapter2.rtf => chapter2.tex (94%)  
rename chapter3.rtf => chapter3.tex (95%)
```

This creates a new snapshot that points to the last commits on both branches

HEAD

chapter1.tex
chapter2.tex
chapter3.tex
chapter4.rtf

Git stores a “repository” as a
graph of snapshots

