

Version Control with Git and GitHub

Ben Winjum

OARC

February 16, 2023

What is version control
and why should I use it?



FINAL.doc!





FINAL_rev.2.doc

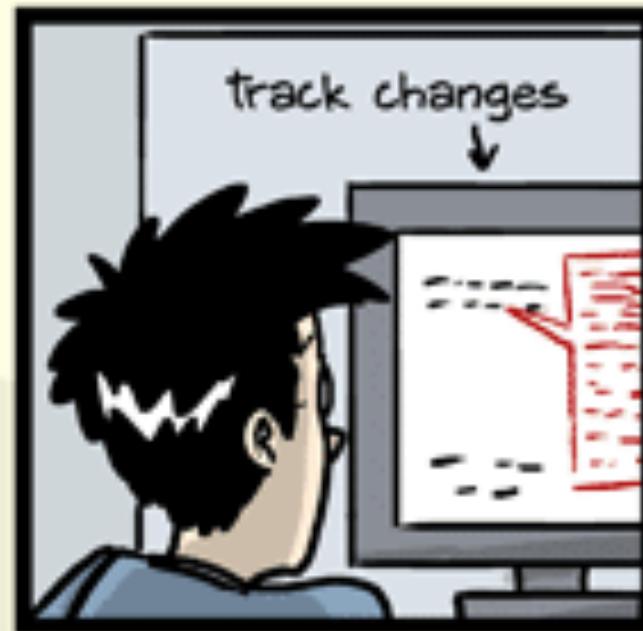


FINAL_rev.6.COMMENTS.doc





FINAL_rev.8.comments5.
CORRECTIONS.doc



FINAL_rev.18.comments7.
corrections9.MORE.30.doc





↑
FINAL_rev.22.comments49.
corrections.10.#@\$%WHYDID
ICOMETOGRADSCHOOL????.doc



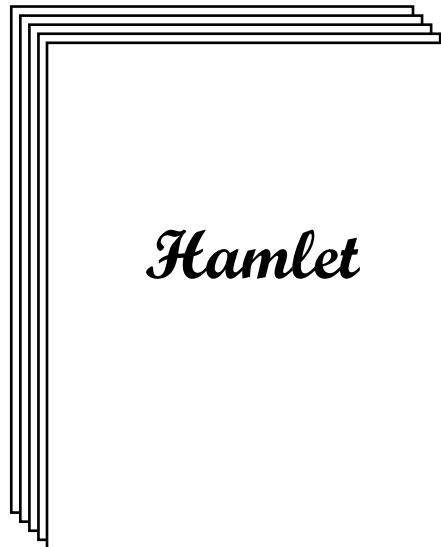
FINAL_rev.22.comments49.
corrections.10.#@\$%WHYDID
ICOMETOGRADSCHOOL?????.doc

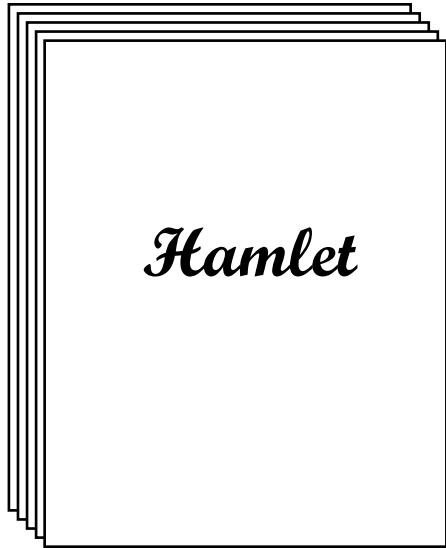
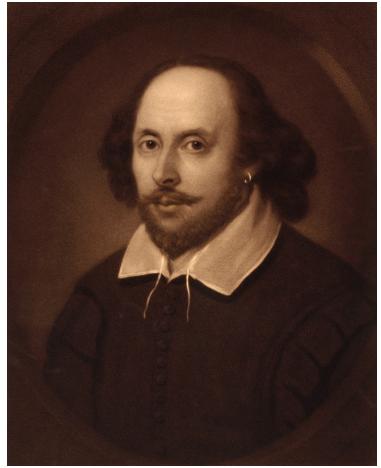
Version Control

Save the current state (FINAL.doc)

and the entire history of your changes

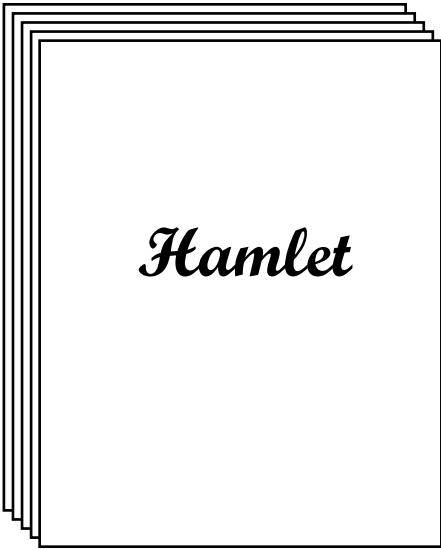
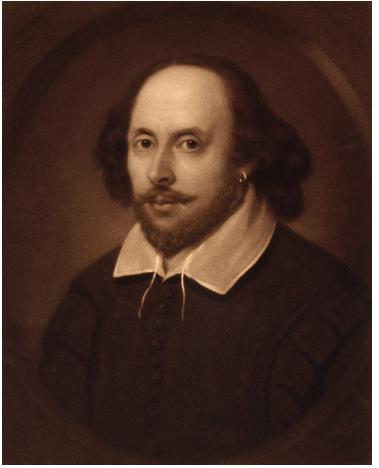
Example: Shakespeare writing Hamlet





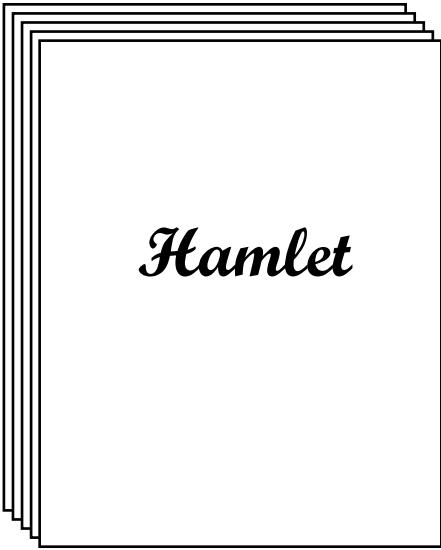
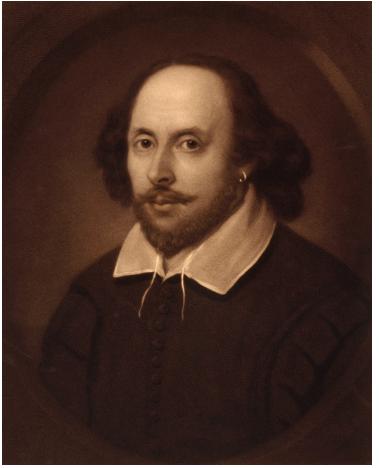
Act III, Scene I

Hamlet:



Act III, Scene I

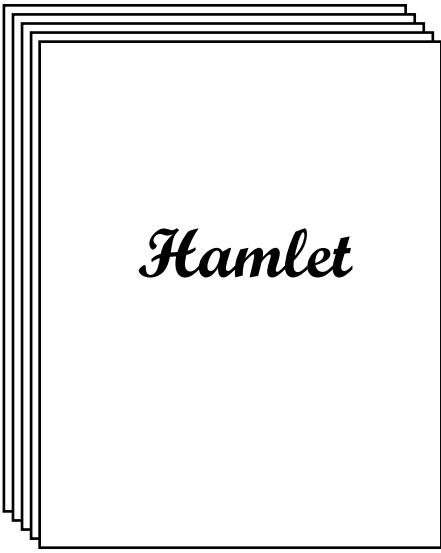
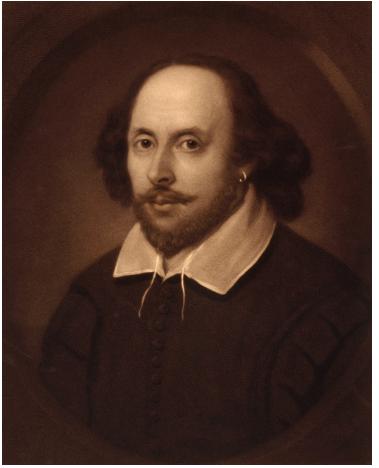
Hamlet:
To be or not to be,
It makes one wonder
About the skies and
The seas, and oysters
And things....



Act III, Scene I

Hamlet:

To be or not to be,
That is the question
It makes one wonder
About the skies and
The seas, and oysters
And things....



Act III, Scene I

Hamlet:

To be or not to be,

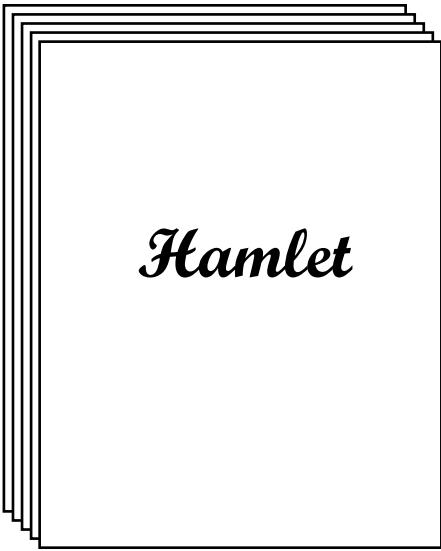
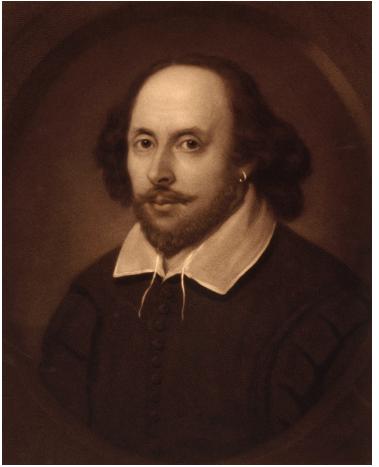
That is the question

It-**That** makes one wonder

About the skies and

The seas, and oysters

And things....



Act III, Scene I

Hamlet:

To be or not to be,

[That is the question](#)

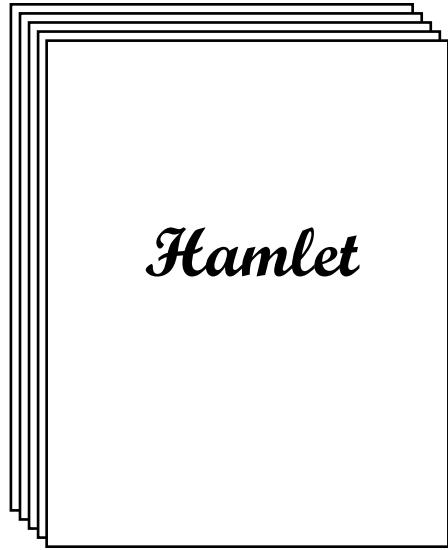
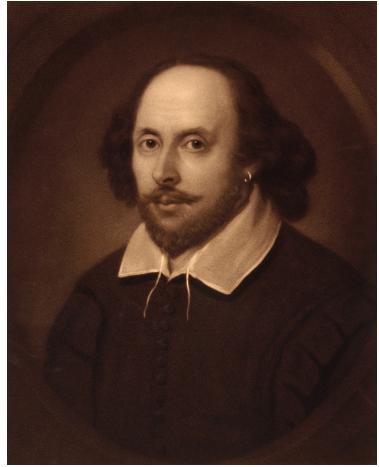
~~It~~-[That](#) makes one wonder

About the skies and

The seas of outrageous fortunes

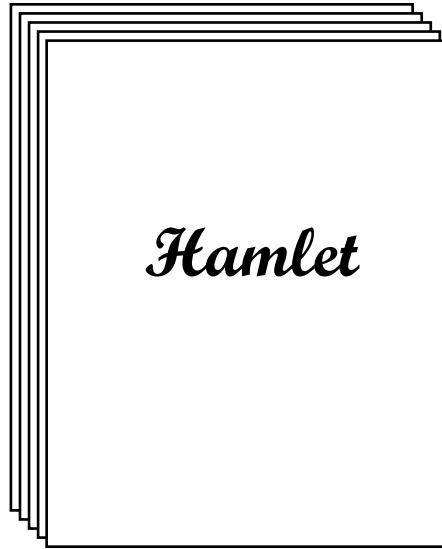
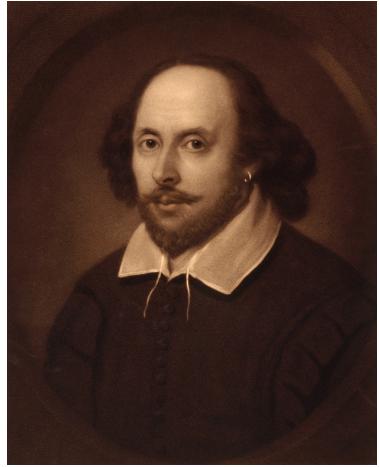
~~, and oysters~~

~~And things....~~



Act III, Scene I

Hamlet:
To be or not to be,
That is the question
That makes one wonder
About the skies and
The seas of outrageous fortunes

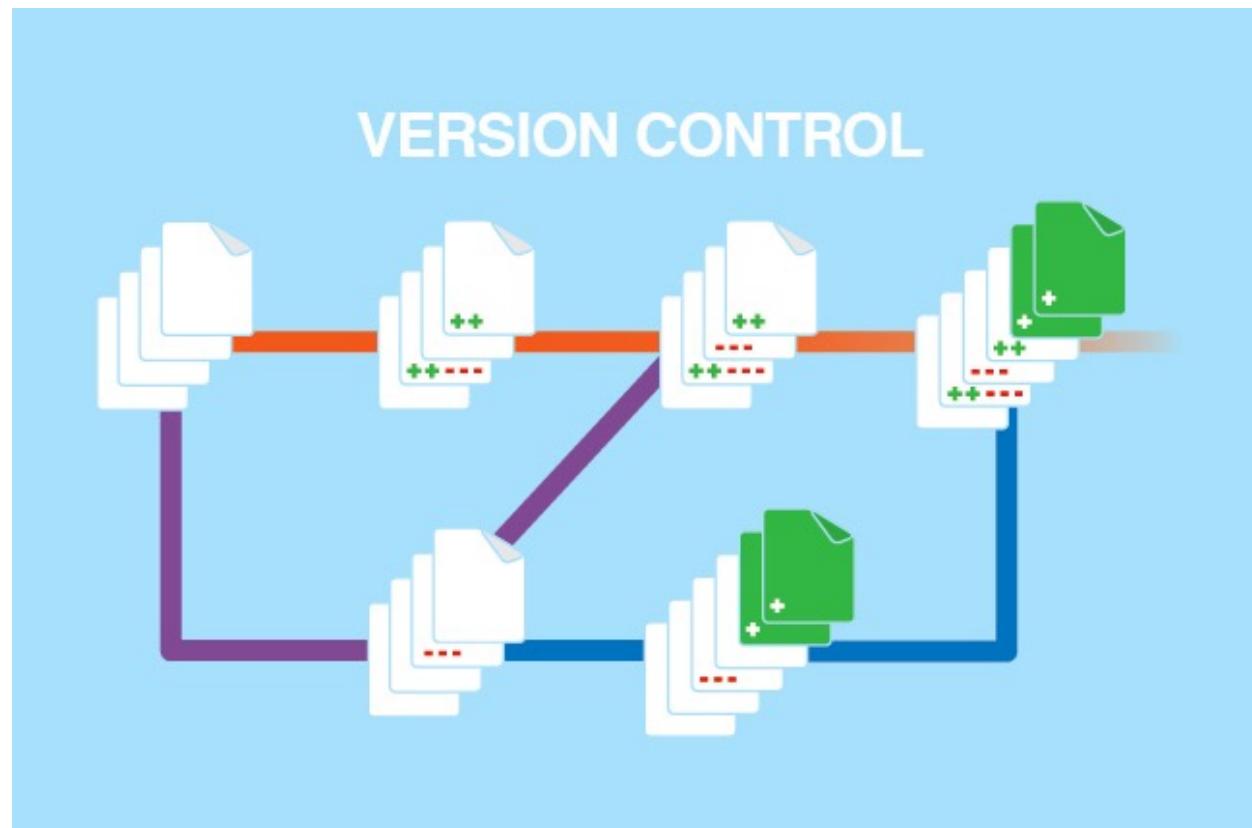


If only Shakespeare had
been able to use git....

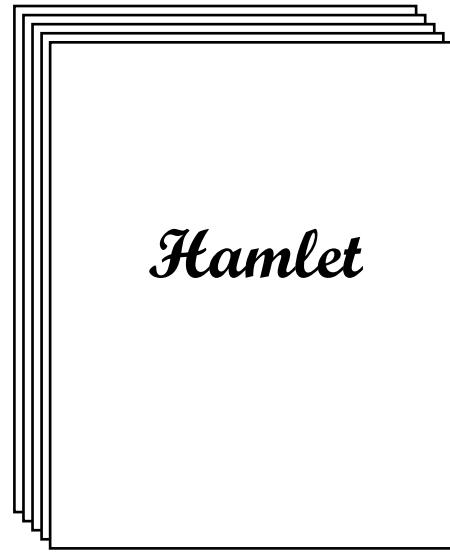
Act III, Scene I

Hamlet:
To be or not to be,
That is the question
That makes one wonder
About the skies and
The seas of outrageous fortunes

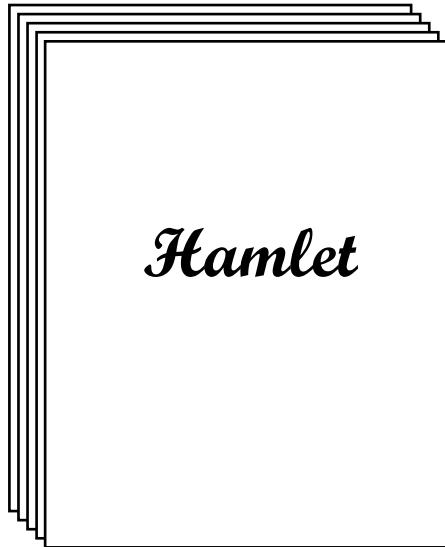
Git is a Version Control System



Git repository:
the collection of files

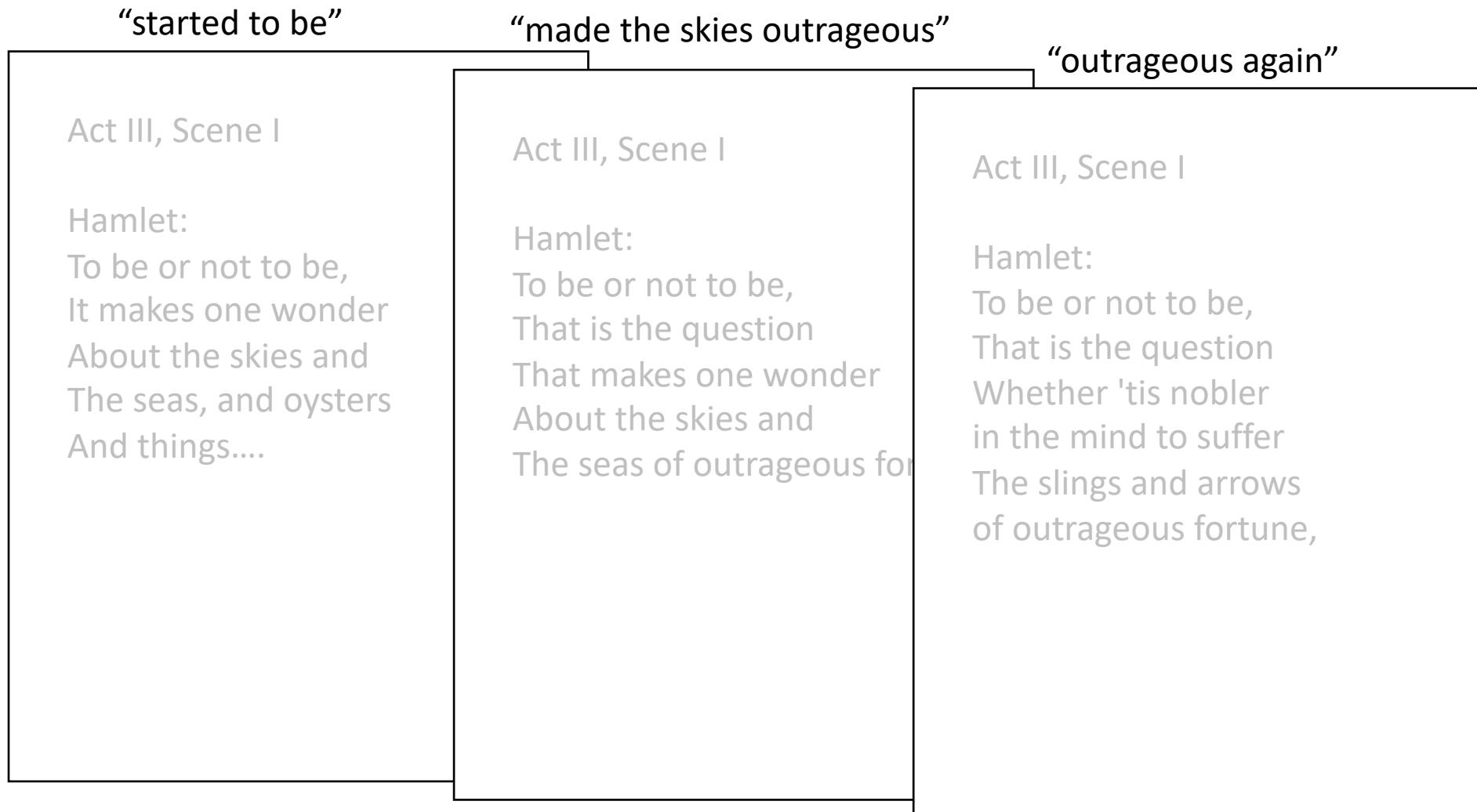


Git repository:
the collection of files
and
the change history

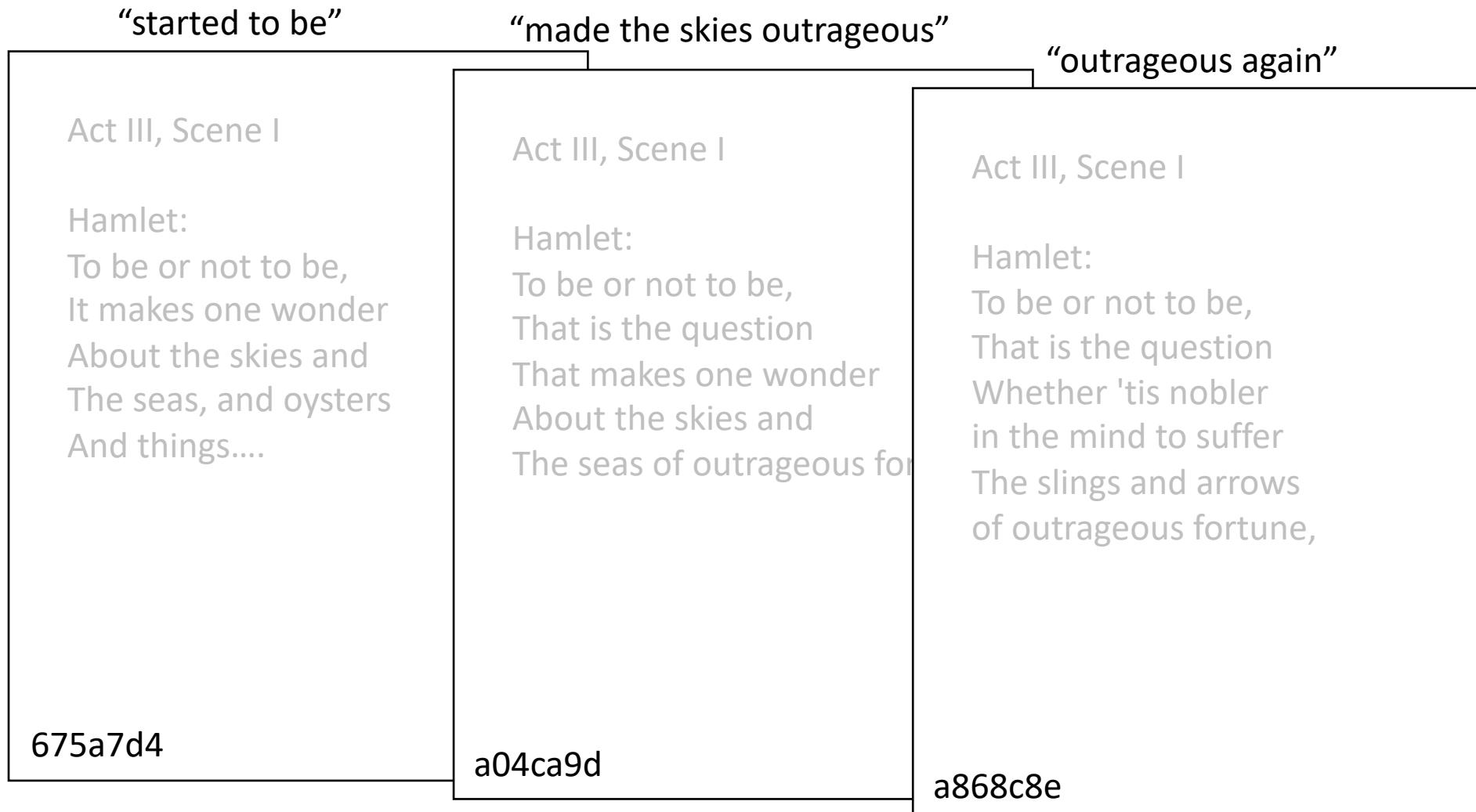


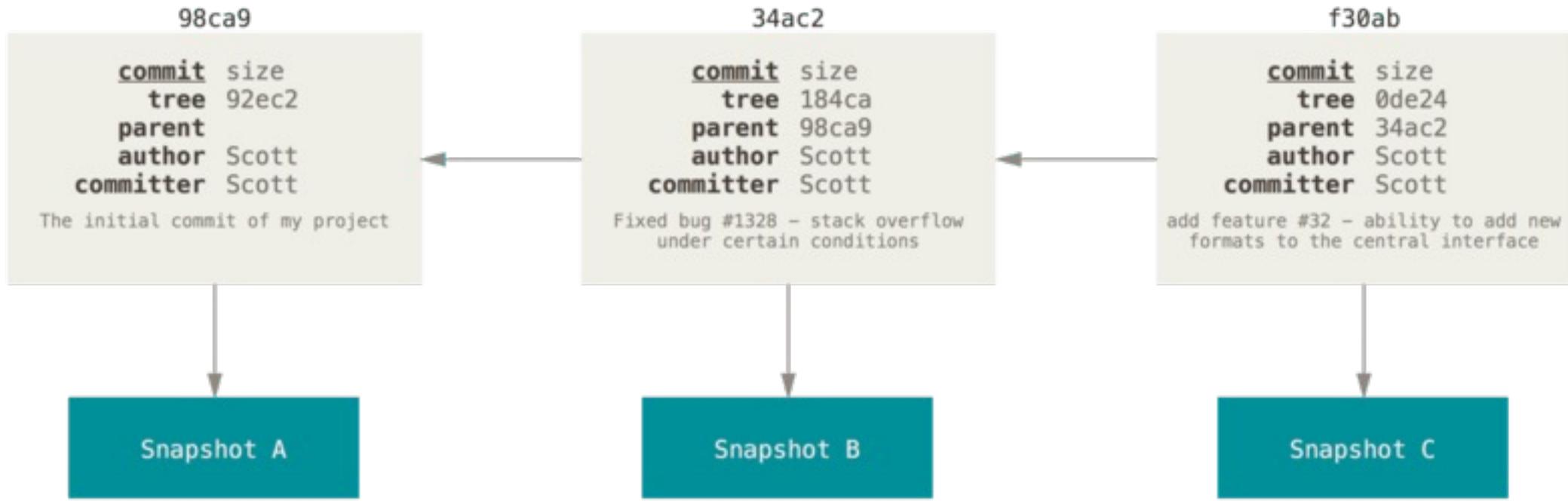
↑	@@ -2,7 +2,7 @@ Act III, Scene I
2	2
3 Hamlet:	3 Hamlet:
4 To be or not to be,	4 To be or not to be,
5 - It makes one wonder	5 + That is the question
	6 + That makes one wonder
6 About the skies and	7 About the skies and
7 - The seas, and oysters	8 + The seas of outrageous fortunes
8 - And things....	

Update the repository by making a “commit” with a message describing the change



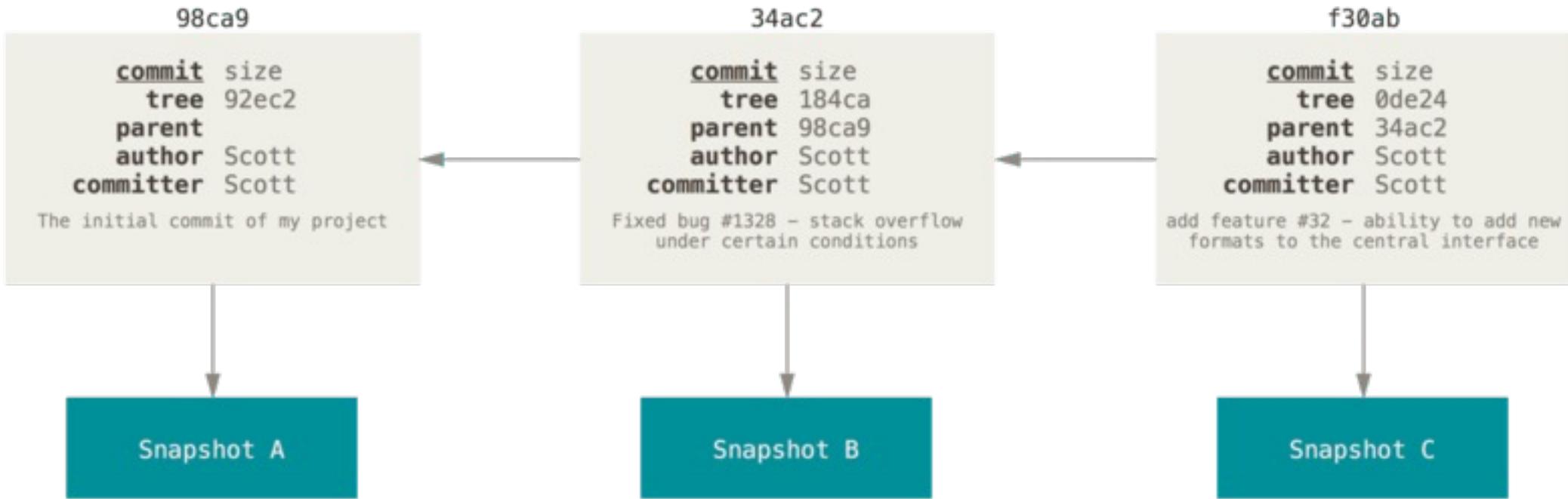
Update the repository by making a “commit” with a message describing the change





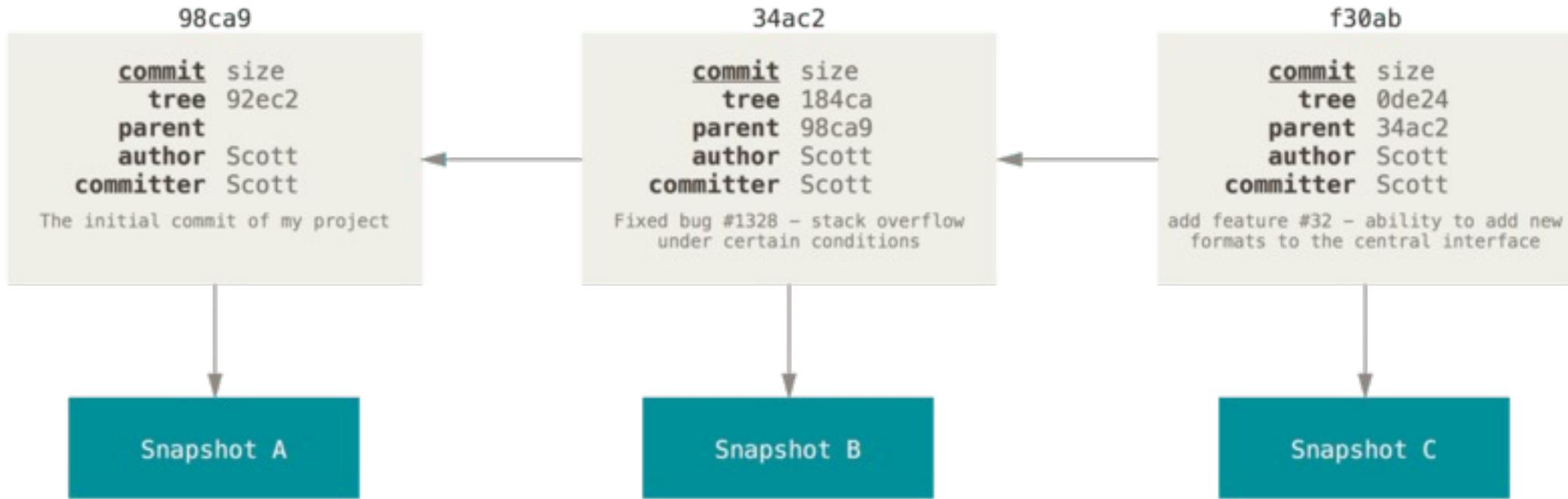
Images taken from the Pro Git book -- freely available online and recommended for further reading
<https://git-scm.com/book/en/v2>

But wait.... Shakespeare isn't the author here....

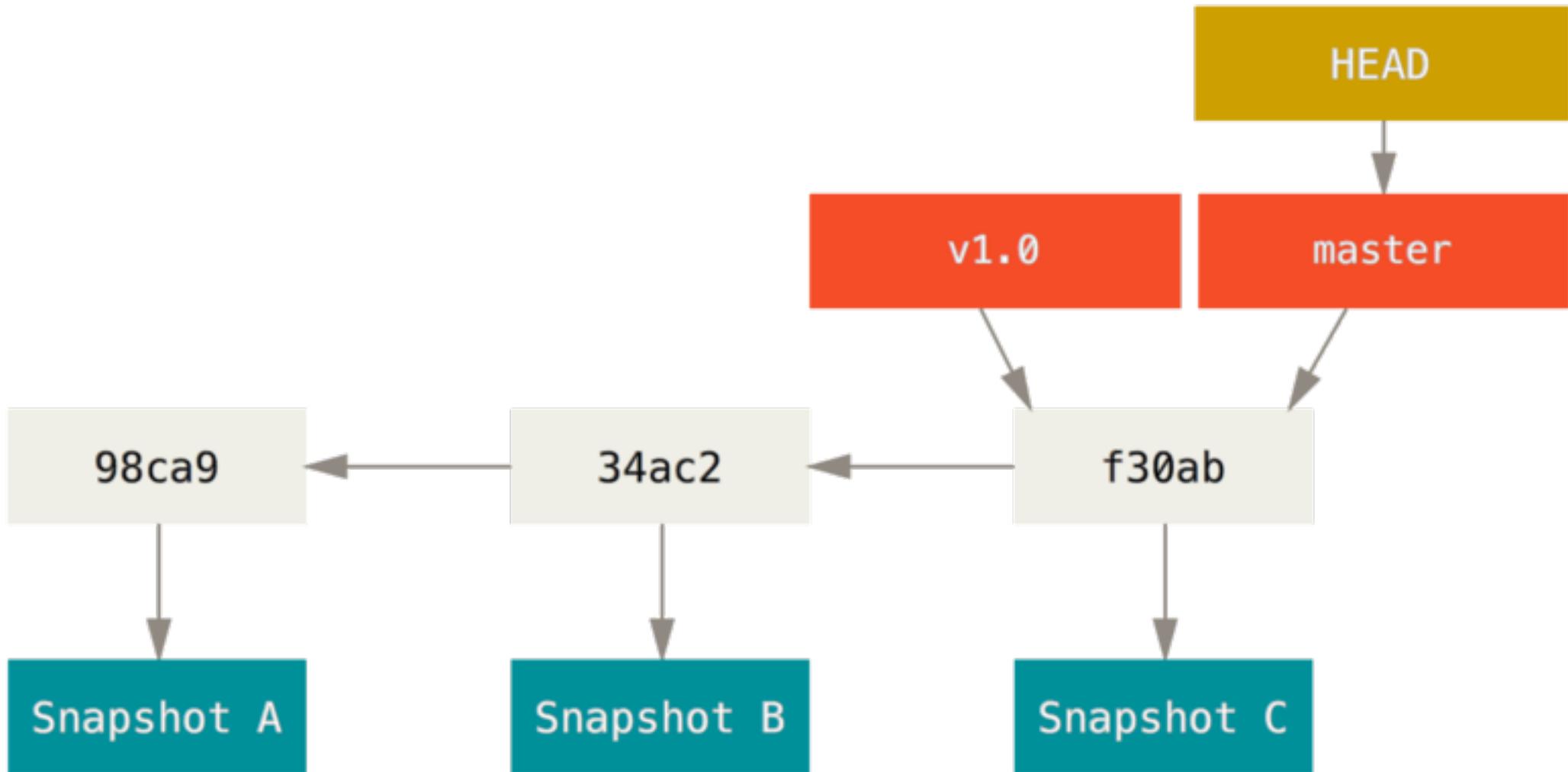


Images taken from the Pro Git book -- freely available online and recommended for further reading
<https://git-scm.com/book/en/v2>

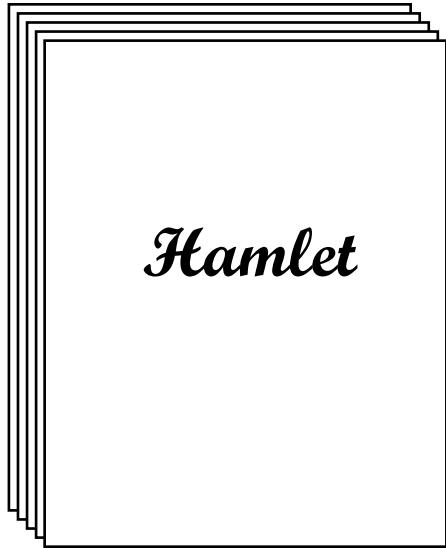
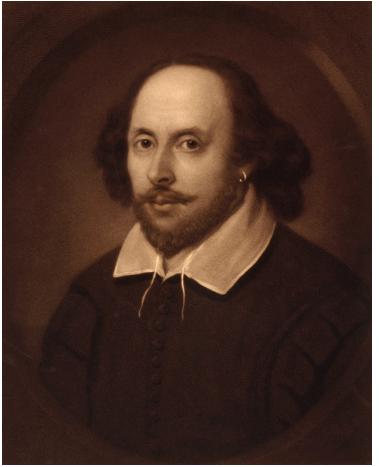
But wait.... Shakespeare isn't the author here....
Best practice: don't commit as an unrecognized author



Images taken from the Pro Git book -- freely available online and recommended for further reading
<https://git-scm.com/book/en/v2>

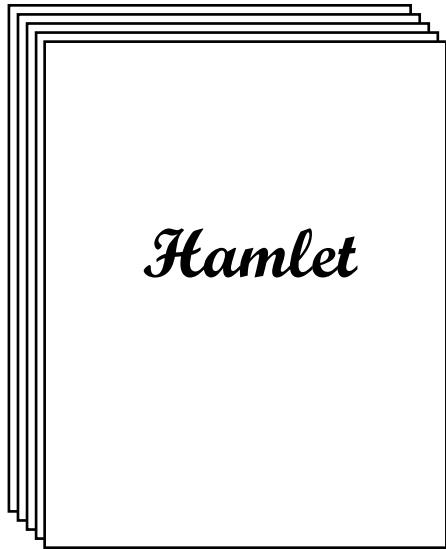
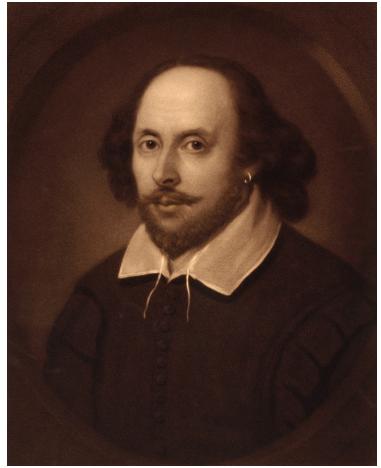


Images taken from the Pro Git book -- freely available online and recommended for further reading
<https://git-scm.com/book/en/v2>



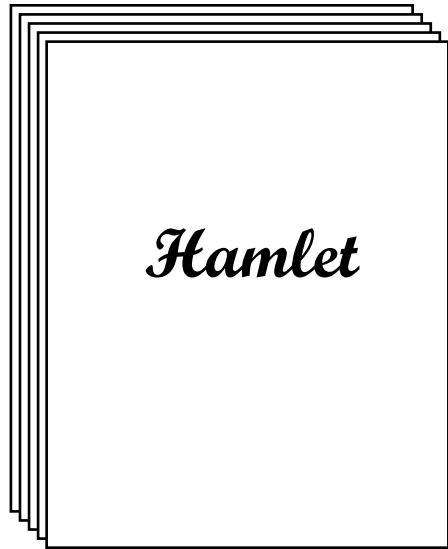
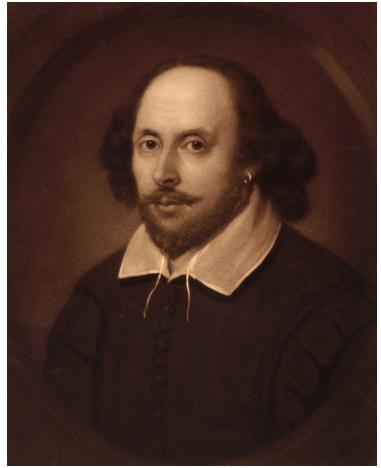
If only Shakespeare had
been able to use git....





If only Shakespeare had
been able to use git....



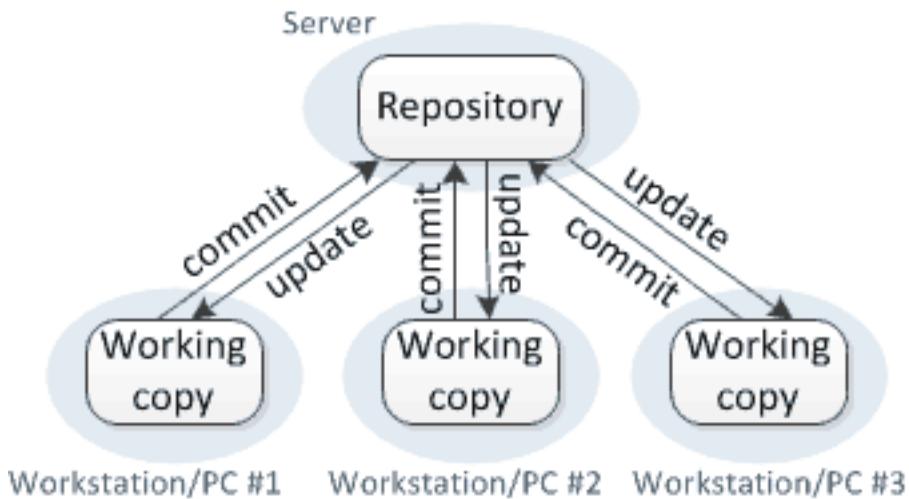


git is used for *collaboration!*

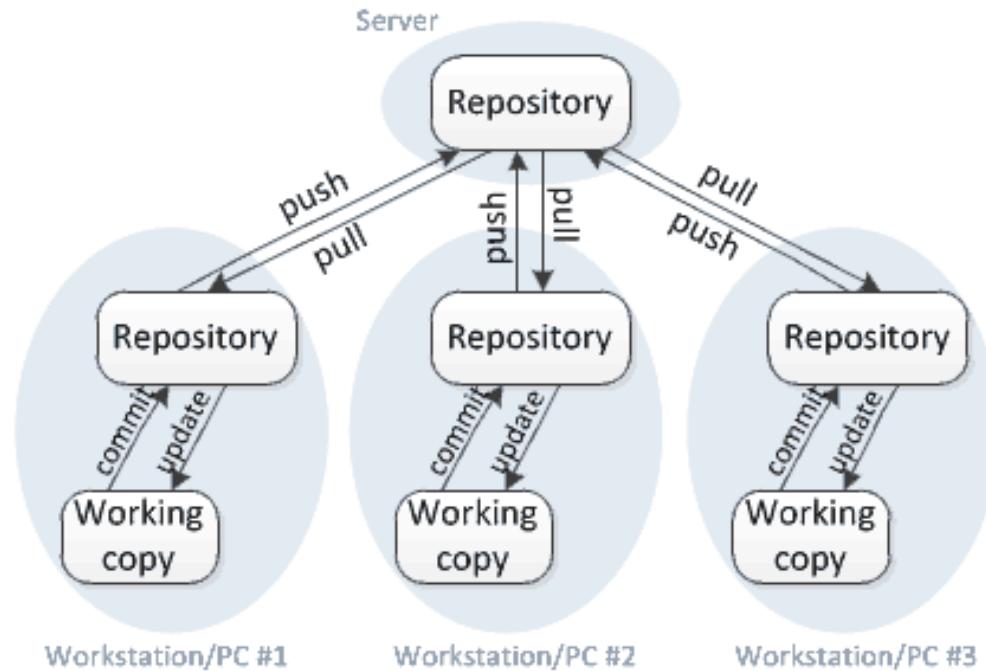


Git is a **Distributed** Version Control System

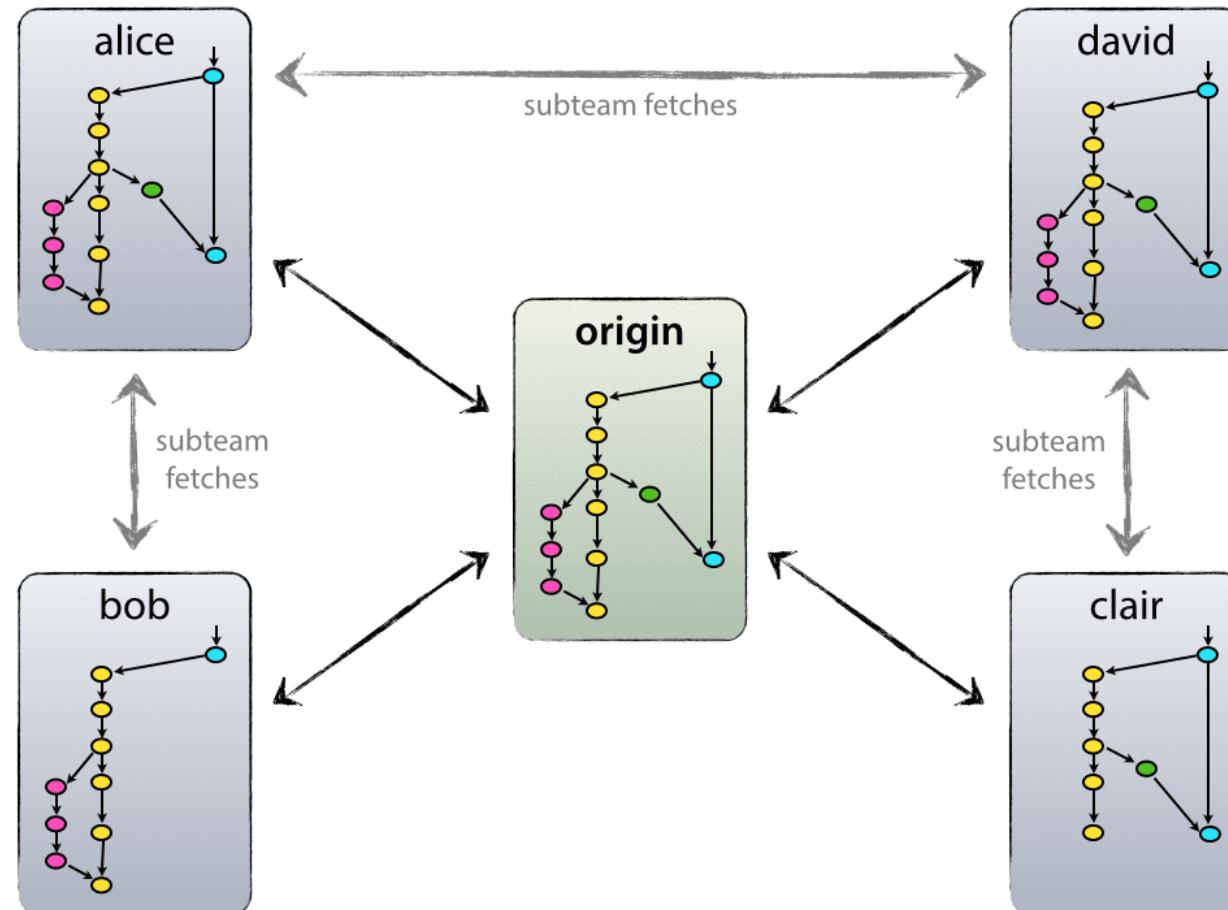
Centralized version control

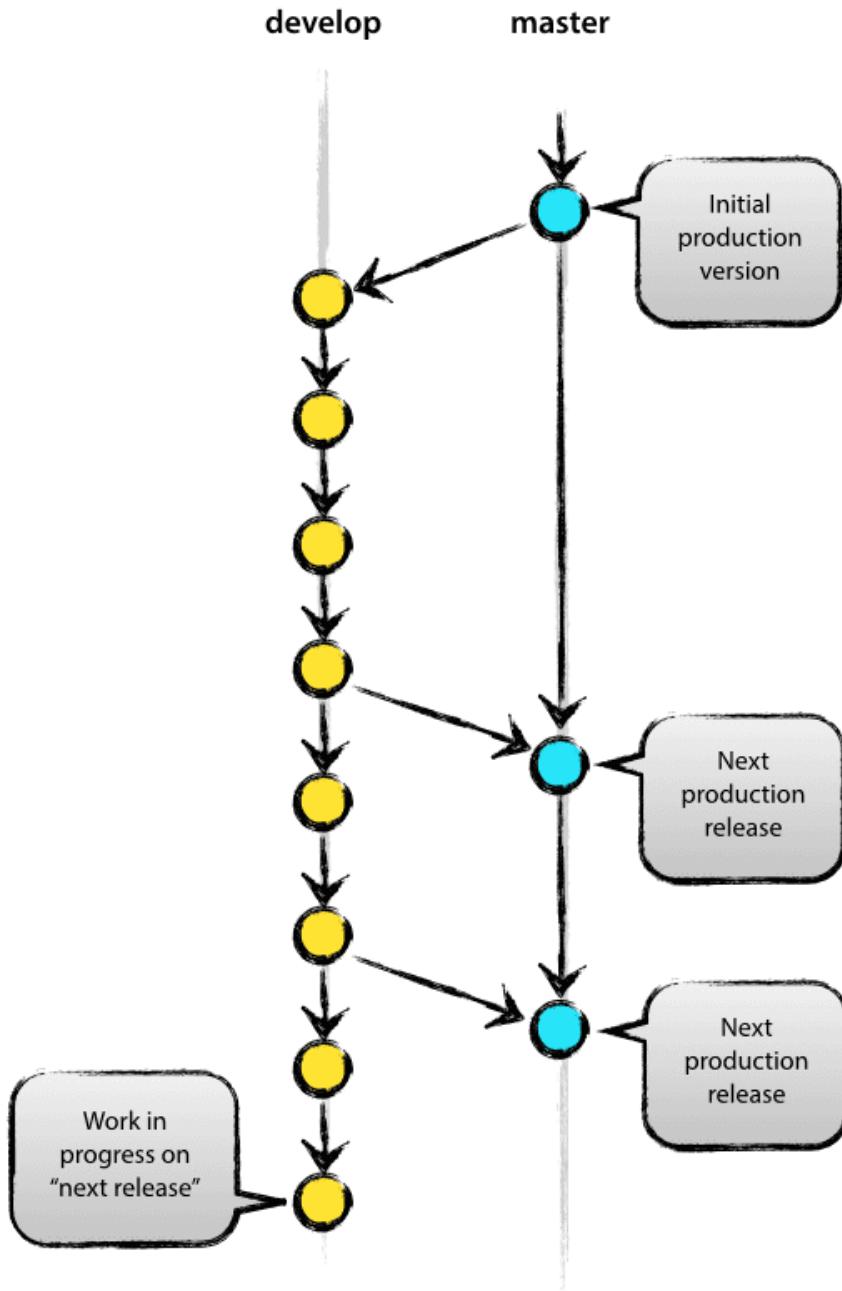


Distributed version control



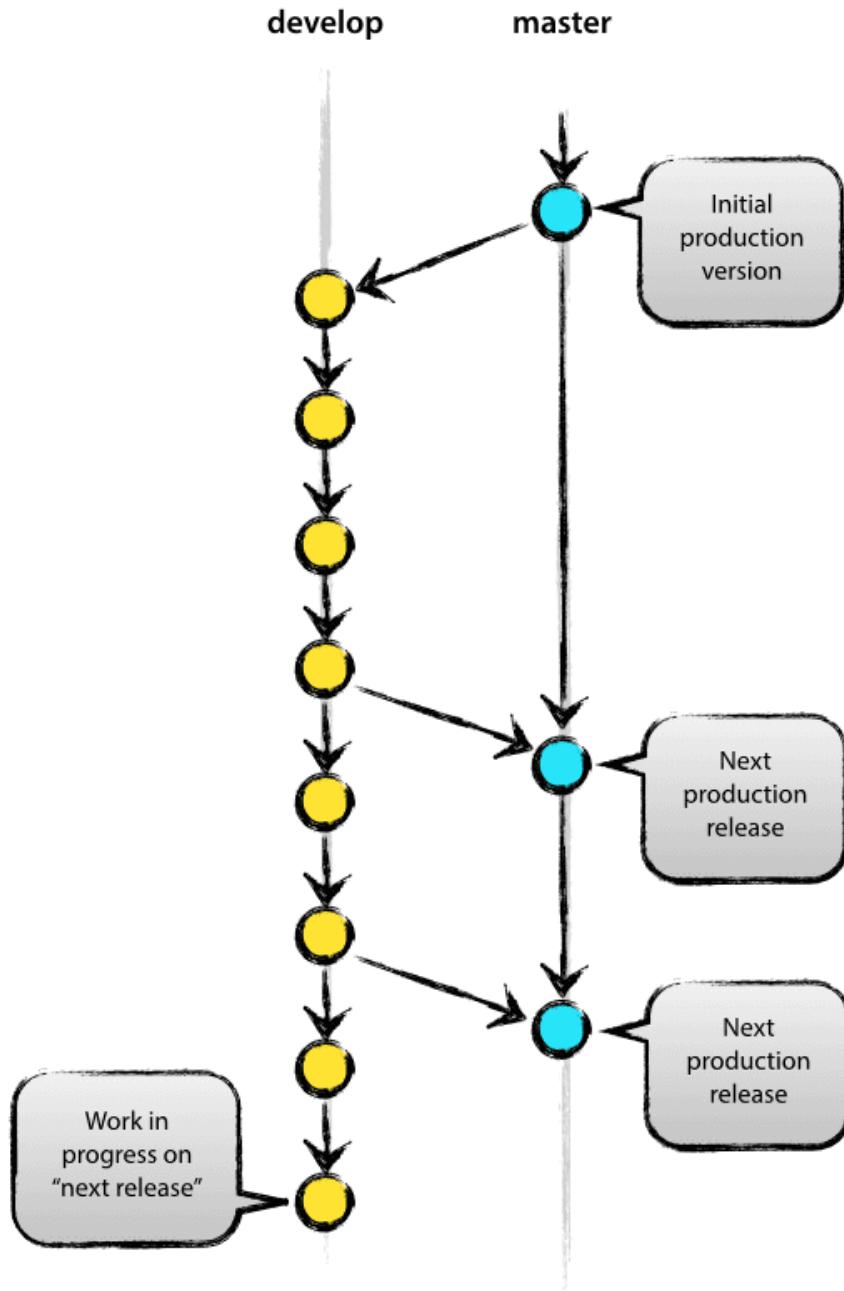
Git is a **Distributed** Version Control System





Development with git can occur in "branches"

These are separate but related lines of modifications

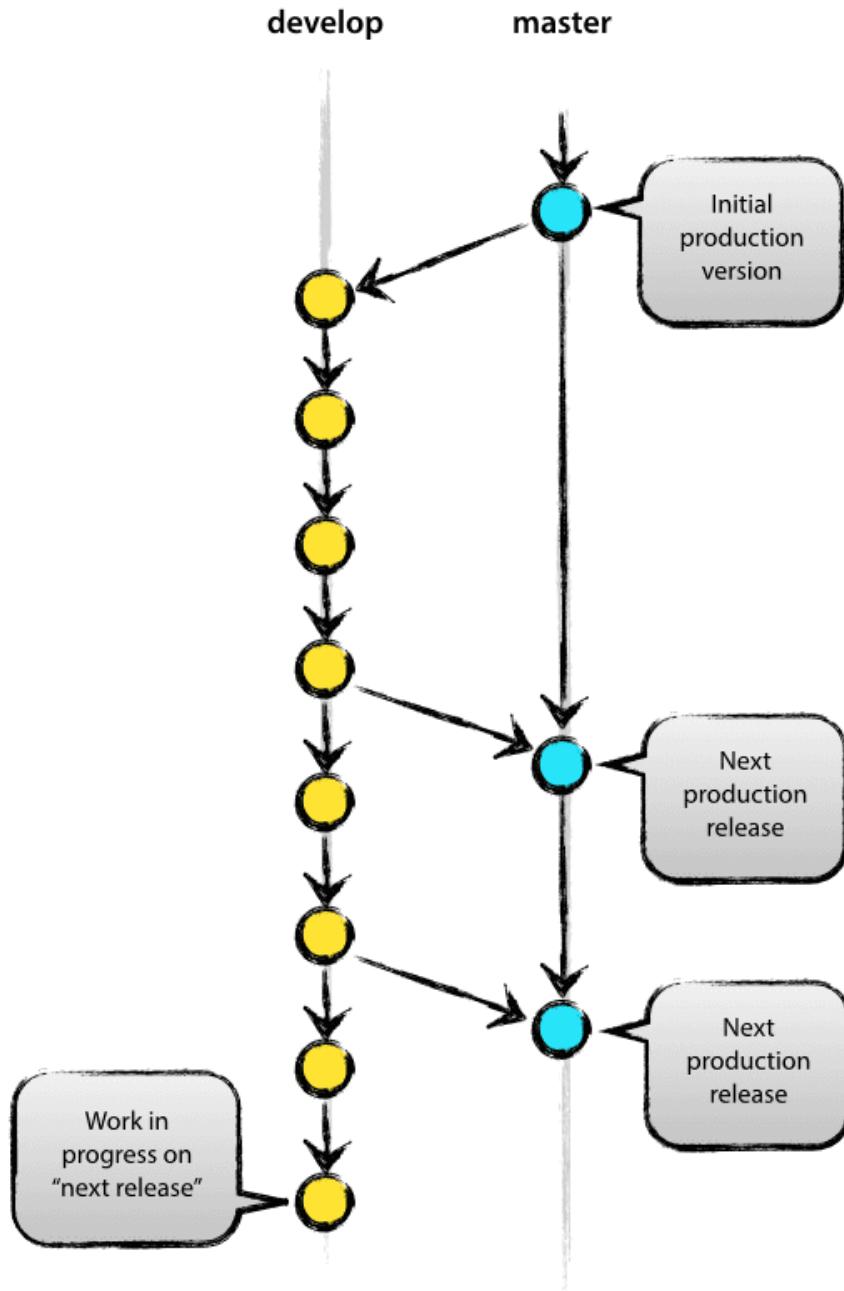


Development with git can occur in "branches"

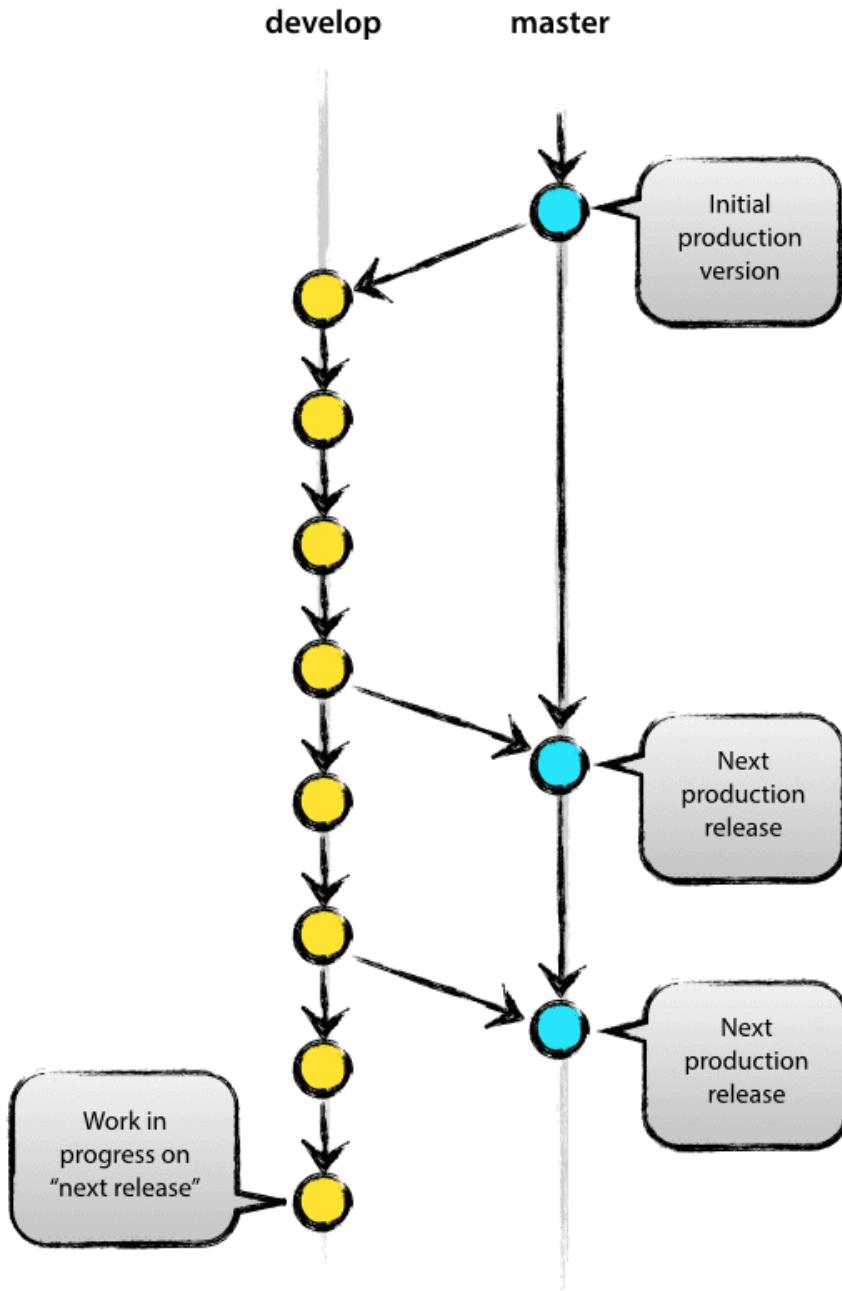
These are separate but related lines of modifications

Example:

- Develop a new feature on a feature branch
- When done, “merge” the feature branch with the main branch



Say a collaborator works on
a new feature

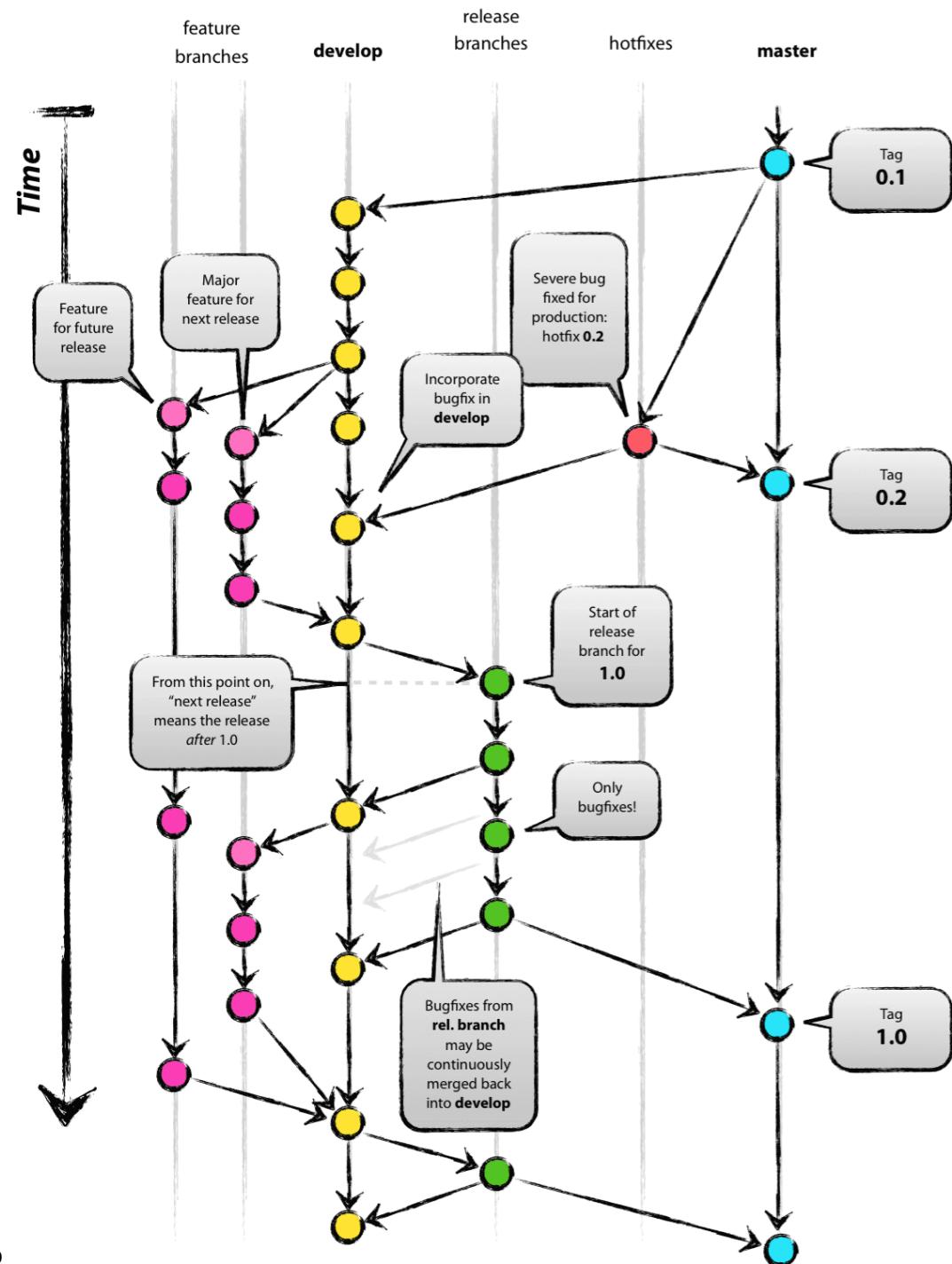


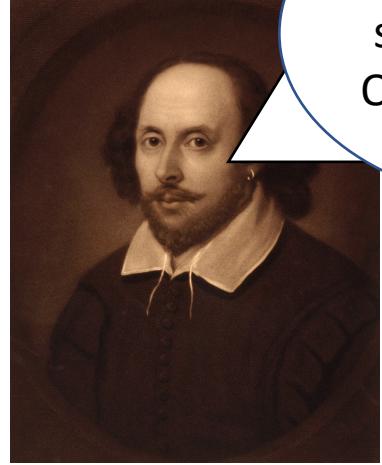
Say a collaborator works on a new feature

When they want to merge it with the main branch

-> “Hey, repo-owner, merge this into the main branch”

-> pull request



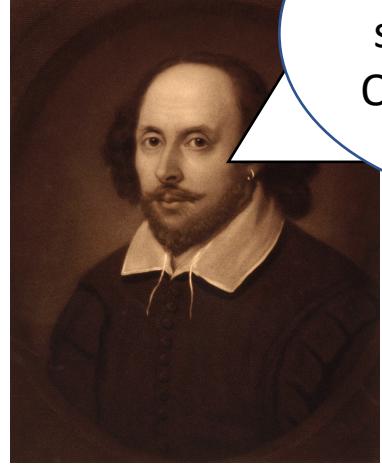


I wish I could get
some input from
Queen Elizabeth...

Hamlet

git is used for *collaboration!*



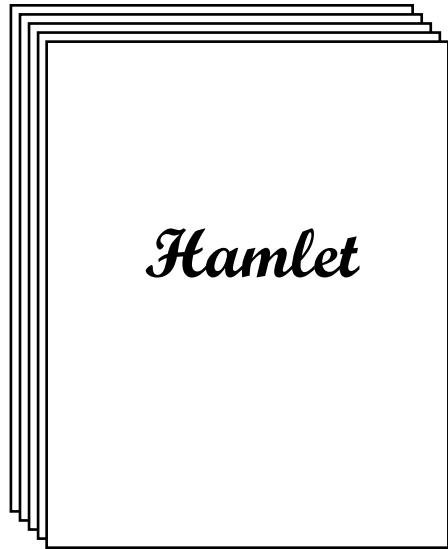
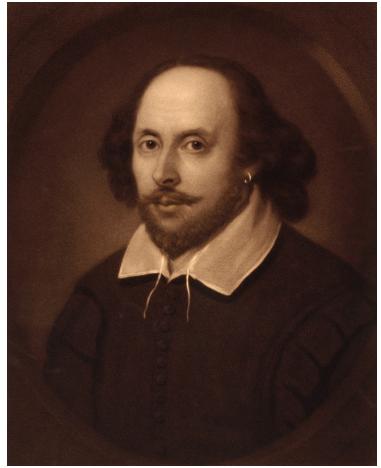


I wish I could get
some input from
Queen Elizabeth...

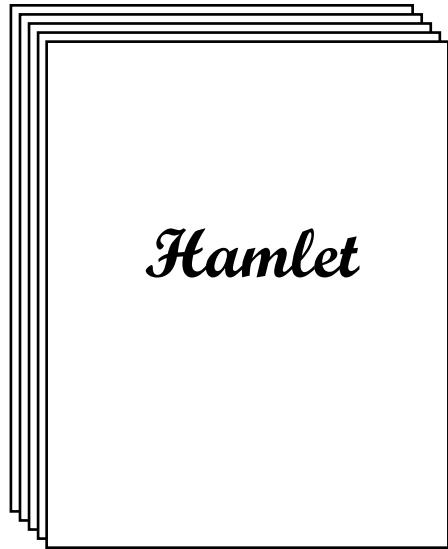
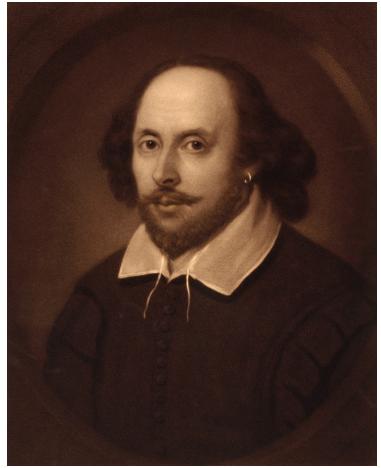
Hamlet

GitHub (and GitLab and Bitbucket)
are git-based platforms that
combine git with a rich set of tools
for development and collaboration



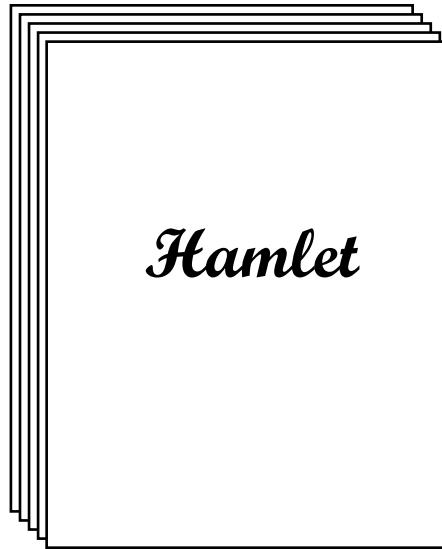
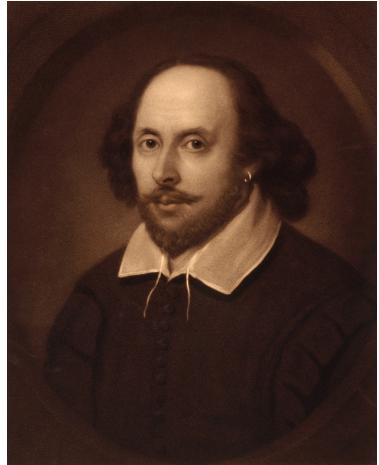


Willie, you put your
social security
number on the title
page



github.com is a publicly accessible platform



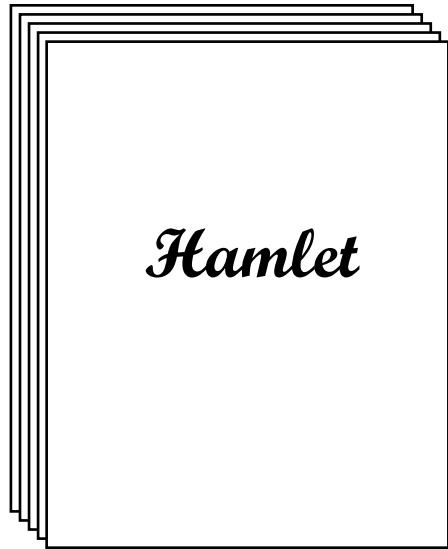
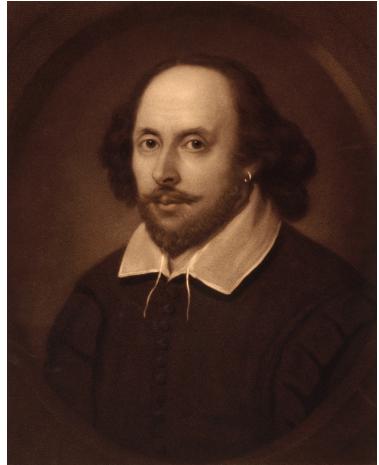


Best practices:

- Use Two-Factor Authentication
- Don't put passwords, tokens, ssh keys, etc in your repo
- If you have sensitive files, use git-crypt, sops, or etc to encrypt your files

github.com is a publicly accessible platform





Best practices:

- Use `.gitignore` to specify files that should not be committed
- Don't commit your local configuration files

github.com is a publicly accessible platform



For our purposes, we will be testing out these concepts with very simple files

First things first

- These slides are available on GitHub, as is a link to get us into a common computing environment
 - Go to https://github.com/benjum/OARC_Git_Workshop
- Create a GitHub account if you don't have one
 - [Github.com](https://github.com)

We will interact with git via the Terminal

- Key shell commands to know
 - `pwd`
 - Print the name of the directory you are currently in
 - `cd <dirname>`
 - Go to the directory named `<dirname>`
 - “cd” is for change directory
 - `ls`
 - List the files and folder names in your current directory

First things first

- Git commands are written as `git verb options`
- Example: setting up our initial configuration

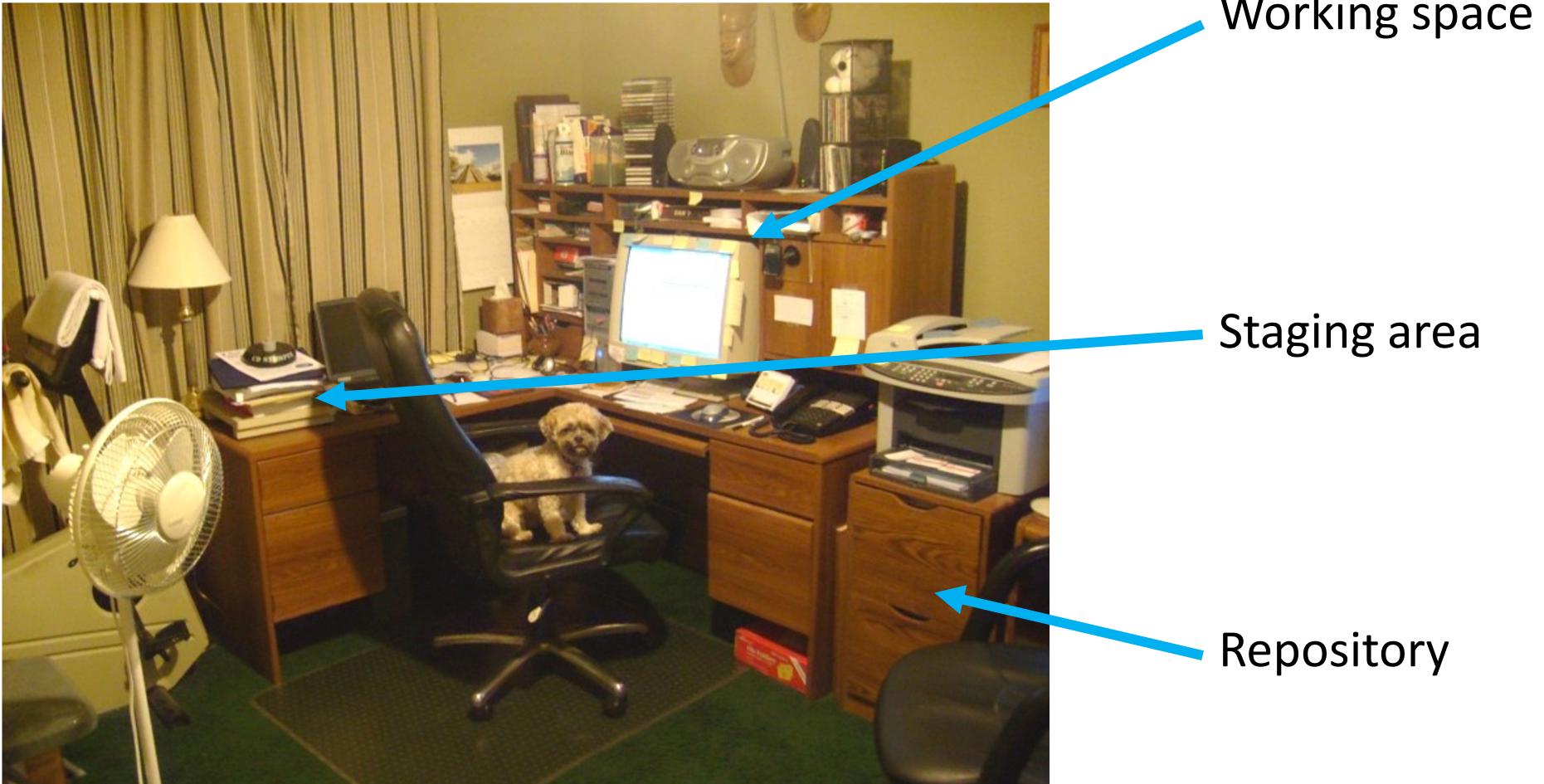
```
$ git config --global user.name "first last"  
$ git config --global user.email "username@ucla.edu"
```

```
$ git config --list
```

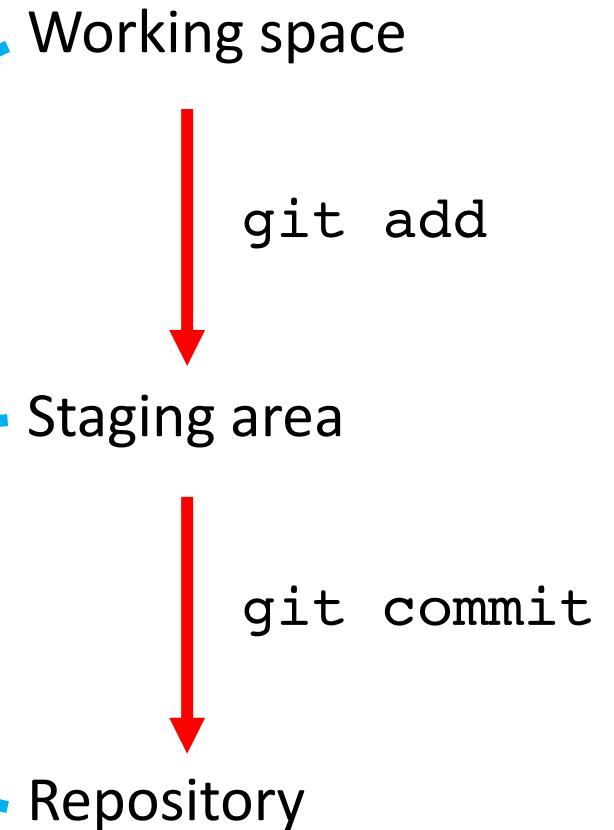
A few git commands

- `git clone repository_name`
 - Copy files from repository_name
- `git config`
 - See next slide
- `git add .`
 - Yes, the period is important!
 - Adds all changes to your staging area
- `git commit -m 'message text here'`
 - Moves changes from staging area to your local repository
 - In a terminal in our Jupyter environment, if you execute git commit without the message and wind up in a weird window, try typing “:wq!” and hit enter
- `git push`
 - Move changes from your local directory to GitHub

Git is like a desk



Git is like a desk



Making a repository

- `git init` is used to initialize a Git repository in the directory in which it is executed
 - `git init dirname` will make a directory called dirname and initialize the repo inside
 - `git init` can be run in subdirectories, but it is redundant and can cause trouble down the road
- `.git` -- Git will store information about the repository in a hidden directory

Adding files and keeping track of versions

- Recording changes
- Checking status
- Making notes of changes
- Ignoring certain changes

git add and git commit

- Adding files to the staging area:
 - `git add file1 file2`
- Moving changes from the staging area into the repository:
 - `git commit -m 'a short message describing the changes'`
- You can add everything at once with: `git add .`
 - But this is not advised in practice
- You can also commit all changes without adding them: `git commit -a`
 - This is also not advised in practice

Best Practice: make good commit messages

	COMMENT	DATE
O	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
O	ENABLED CONFIG FILE PARSING	9 HOURS AGO
O	MISC BUGFIXES	5 HOURS AGO
O	CODE ADDITIONS/EDITS	4 HOURS AGO
O	MORE CODE	4 HOURS AGO
O	HERE HAVE CODE	4 HOURS AGO
O	AAAAAAA	3 HOURS AGO
O	ADKFJSLKDFJSDFKLJ	3 HOURS AGO
O	MY HANDS ARE TYPING WORDS	2 HOURS AGO
O	HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

For Reference

What makes a good commit message?

- <https://cbea.ms/git-commit/>
- <http://tbaggy.com/2008/04/19/a-note-about-git-commit-messages.html>
- https://www.git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project#_commit_guidelines
- <https://github.com/torvalds/subsurface-for-dirk/blob/master/README.md#contributing>
- <http://who-t.blogspot.co.at/2009/12/on-commit-messages.html>
- <https://github.com/erlang/otp/wiki/writing-good-commit-messages>
- <https://github.com/spring-projects/spring-framework/blob/30bce7/CONTRIBUTING.md#format-commit-messages>

Example: The seven rules of a great Git commit message

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain what and why vs. how

git commit for directories

Git tracks files within directories, but not directories

```
$ mkdir newdir  
$ git status  
$ git add newdir  
$ git status
```

If you have files in a directory, you can add them all at once with:

```
$ git add dirname
```

Committing Changes with Git

Suppose you use Git to manage a file named `myfile.txt`. After editing the file, which command(s) below would save the changes to the repository?

- 1) `git commit -m "my recent changes"`
- 2) `git init myfile.txt; git commit -m "my recent changes"`
- 3) `git add myfile.txt; git commit -m "my recent changes"`
- 4) `git commit -m myfile.txt "my recent changes"`

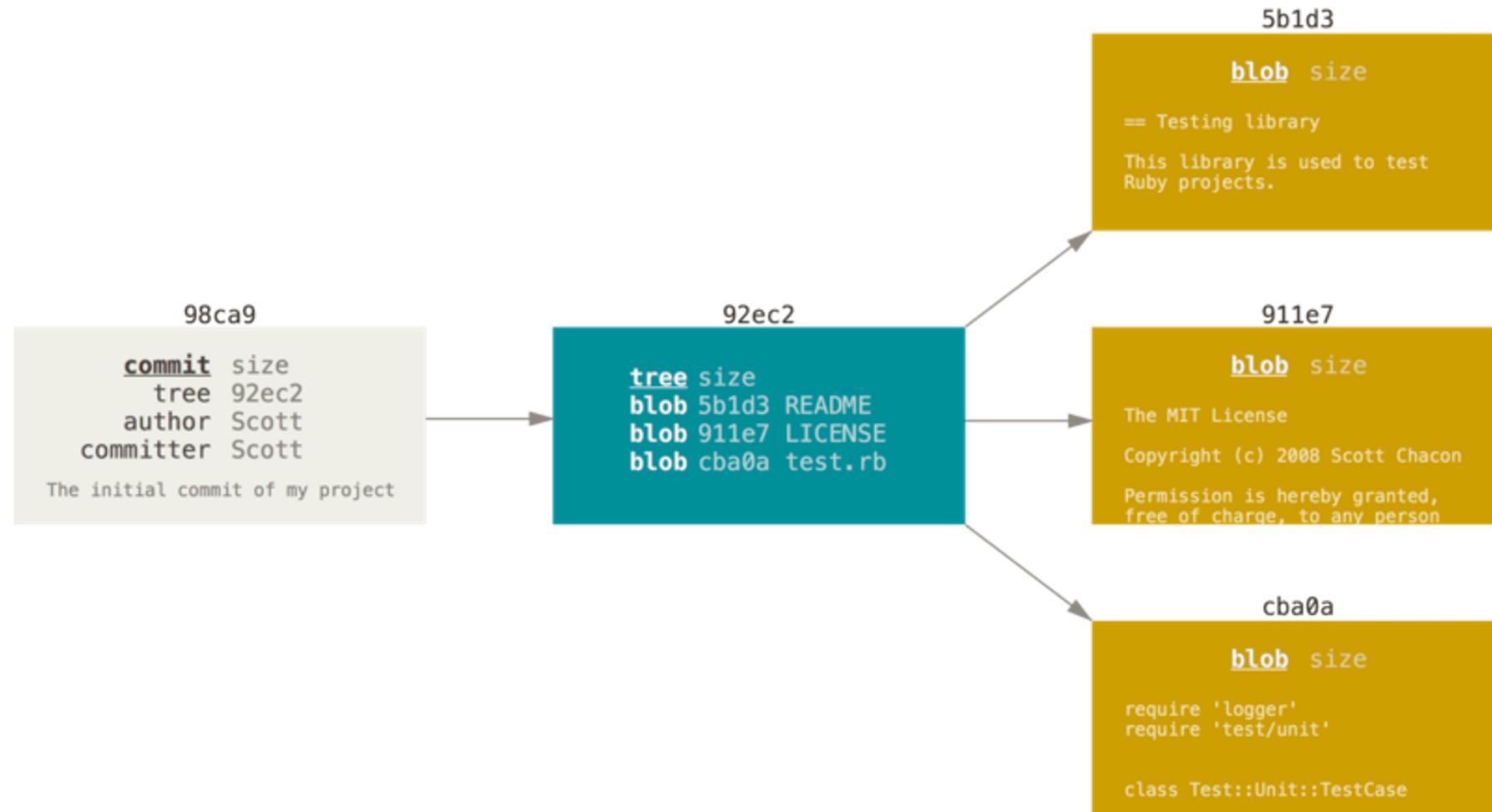
Exercises with `init`, `add`, `commit`

1. Initialize a Git repository
2. Make a new file
3. Commit this file to the repository

Looking at history

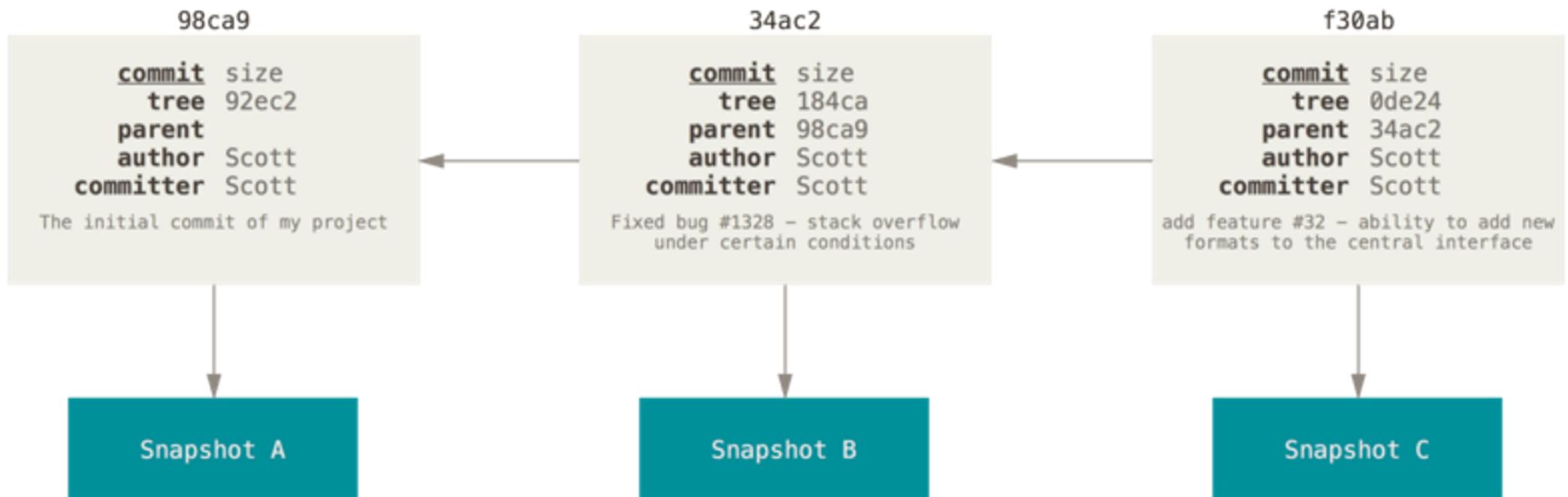
```
$ git log -[N]  
$ git log --oneline  
$ git log --oneline --graph
```

Files and the commit history



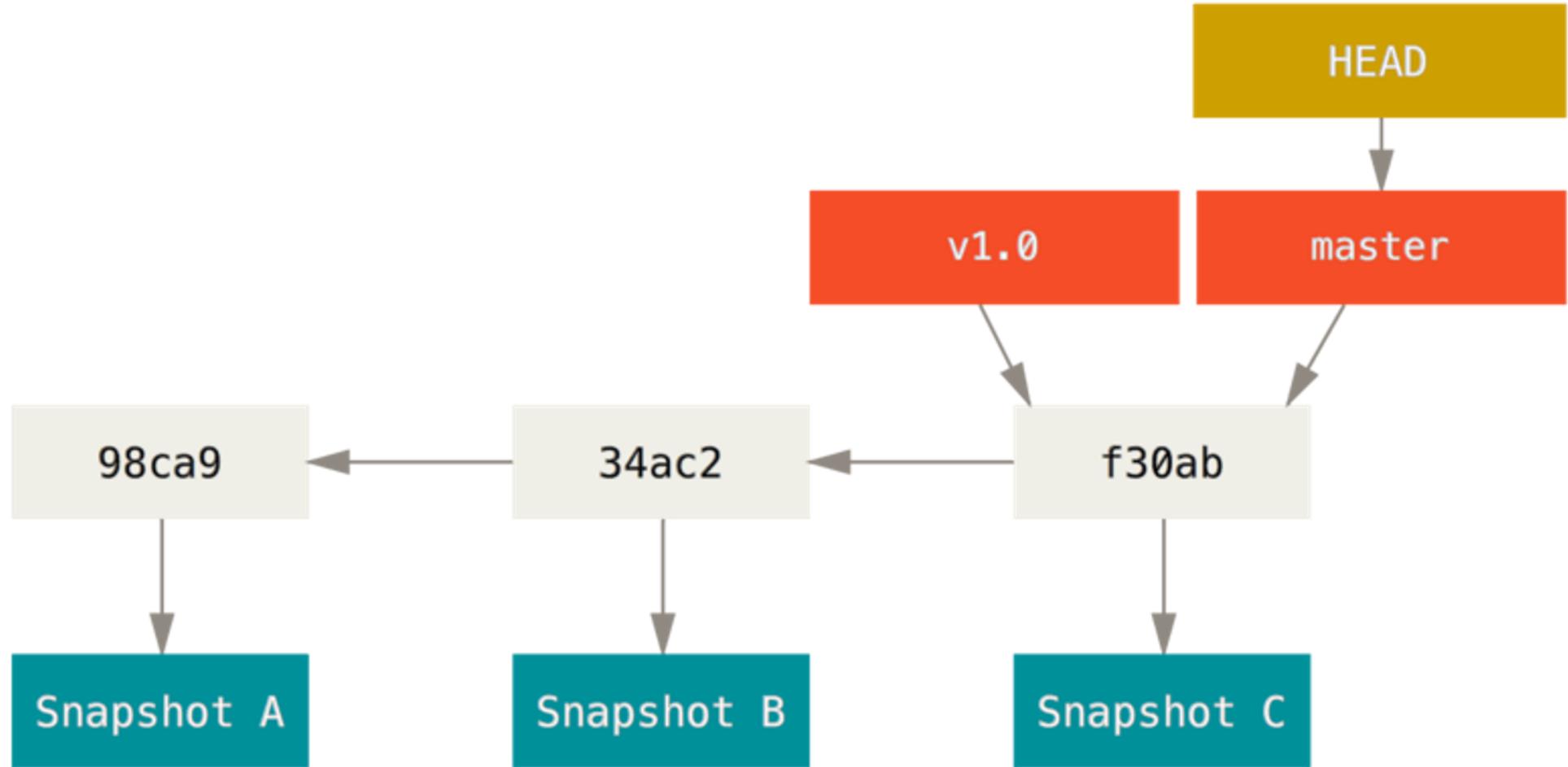
Images taken from the Pro Git book -- freely available online and recommended for further reading
<https://git-scm.com/book/en/v2>

Files and the commit history



Images taken from the Pro Git book -- freely available online and recommended for further reading
<https://git-scm.com/book/en/v2>

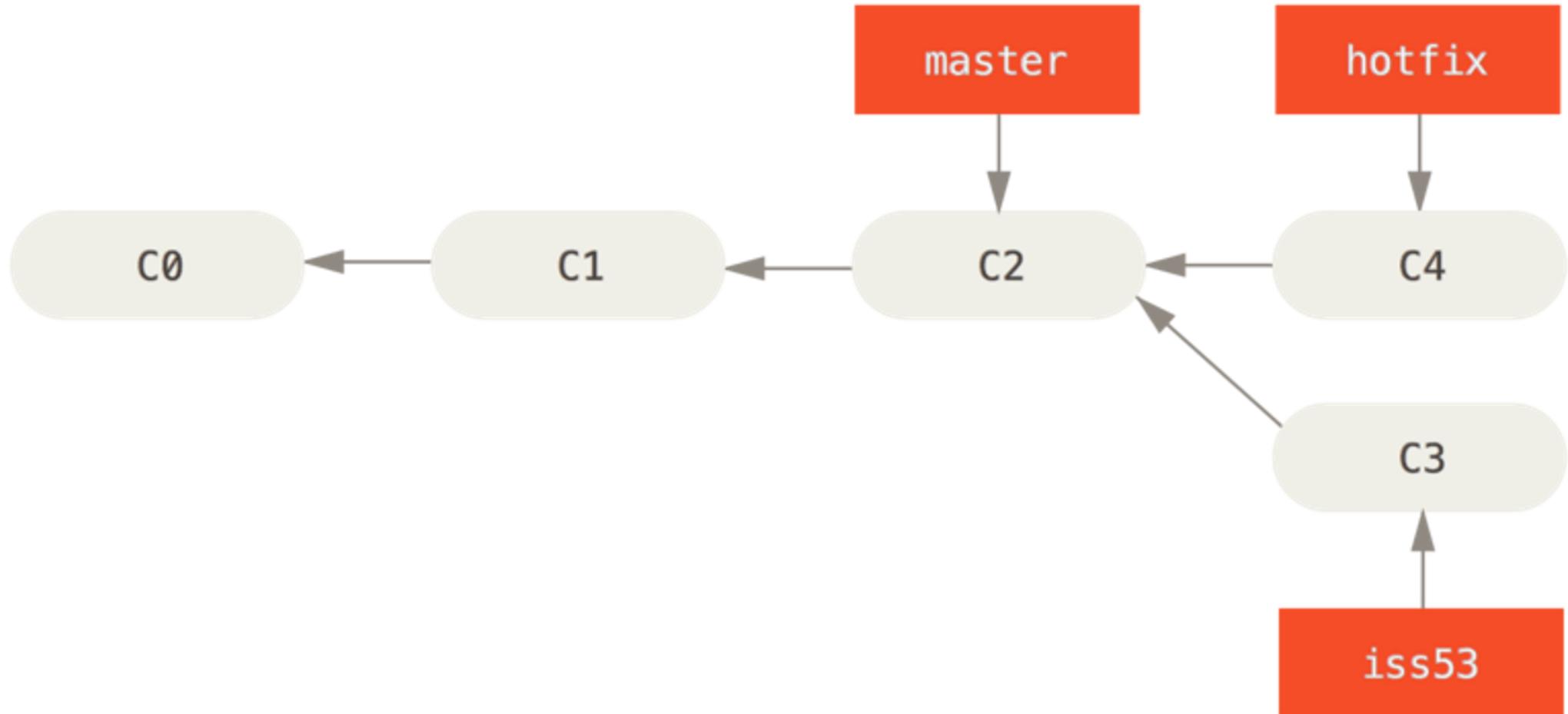
Pointers to commits



The tool for visualizing git actions can be found at:

<http://git-school.github.io/visualizing-git/>

Tracking different development paths via different pointers



Images taken from the Pro Git book -- freely available online and recommended for further reading
<https://git-scm.com/book/en/v2>

Looking at differences

```
$ git diff  
$ git diff --staged  
$ git diff filename
```

Exercises with `add`, `commit`, `diff`, and `log`

1. Make a new file (or files) and/or change files in your working area
2. Display the differences between the files' updated states and the repository's previously committed state
3. Commit your changes
4. View your version history to confirm

Dealing with past history

- Identifying old versions
- Reviewing past changes
- Recovering old versions

Discerning what changed

You can refer to the most recent commit of the working directory by using the identifier HEAD

HEAD~N refers to the Nth parent commit relative to HEAD

```
$ git diff HEAD  
$ git diff HEAD~3 HEAD~1
```

To look at changes that were made in a commit rather than differences between commits, you can use:

```
$ git show
```

git checkout

- Reverting files to a previous state
 - Can be done relative to HEAD
 - Can be specified with 40-digit identifier
 - Can be specified with smaller amount of initial digits
- Beware that the following are different commands!

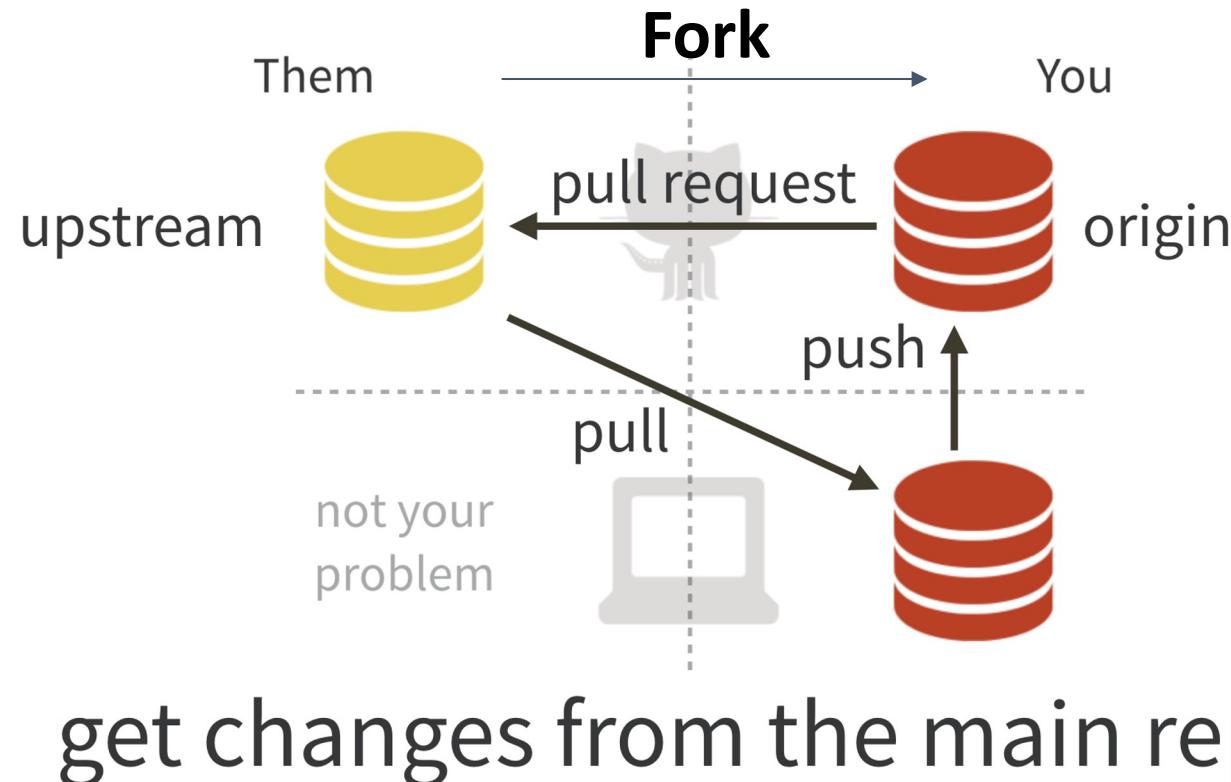
```
$ git checkout HEAD~1 fname.txt  
$ git checkout HEAD~1
```

One reverts fname.txt to a previous state, the other detaches HEAD!

Git + collaboration (using GitHub as example)

- Git already includes the machinery to move work between two repositories
- In practice, a central repository is usually used as a master copy
 - GitHub, BitBucket, GitLab...
- These hosting services also offer tools to facilitate collaboration
 - Web-based – anyone with an internet connection can view the repo
 - Wikis, task management, bug tracking, feature requests

Forks, Pushes, and Pull Requests



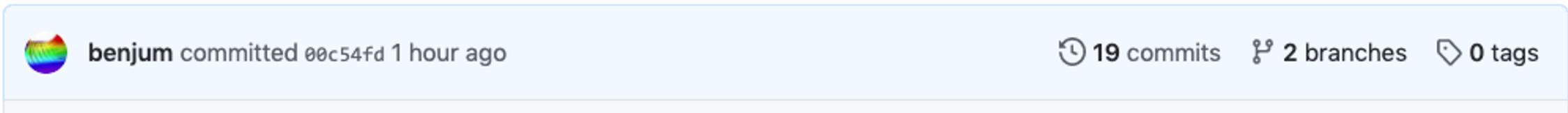
Git + collaboration (using GitHub as example)

- Exchanging changes between repositories

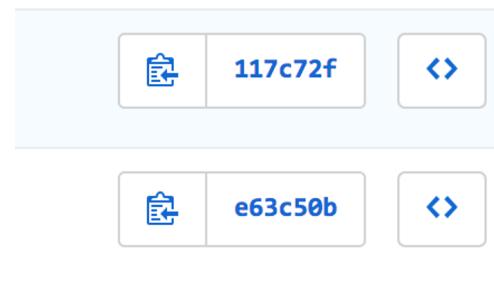
- `git push`
- `git push origin main`
- `git push -u origin main`
- `git pull`
- `git pull origin main`

GitHub exercise

- Look at your new repository on GitHub and find the bar that looks similar to this:



- Click on commits, and then find three buttons per line that look like:



- What do these buttons do when you click them? And how can you do something similar in the terminal?

Collaborative workflow

- The basic collaborative workflow:
 - update your local repo with `git pull origin main`
 - make your changes and stage them with `git add`
 - commit your changes with `git commit -m`
 - upload the changes to GitHub with `git push origin main`
- It is better to make many commits with smaller changes rather than one massive commit with lots of changes
 - Small commits are easier to read and review

Dealing with conflicts

- You will inevitably encounter scenarios in which you make local commits and try to push them to a remote repository, only to discover that a collaborator has updated the repository too
- Git will recognize potential conflicts and refuse to accept a push request
 - Solution: pull changes from the remote repo, merge them locally, and push again

Some final thoughts: If you run into lots of conflicts

- Conflict resolution costs time and effort and can introduce errors if conflicts are not resolved correctly. If you find yourself resolving a lot of conflicts, consider these technical approaches:
 - Pull from upstream more frequently, especially before starting new work
 - Use topic branches to segregate work, merging to main when complete
 - Make smaller more atomic commits
 - Where logically appropriate, break large files into smaller ones so that it is less likely that two authors will alter the same file simultaneously
- Conflicts can also be minimized with project management strategies:
 - Clarify who is responsible for what areas with your collaborators
 - Discuss what order tasks should be carried out in with your collaborators so that tasks expected to change the same lines won't be worked on simultaneously
 - If the conflicts are stylistic churn (e.g. tabs vs. spaces), establish a project convention that is governing and use code style tools (e.g. htmltidy, perltidy, rubocop, etc.) to enforce, if necessary

Final item:
please complete the brief
post-course survey

Any Future Questions:
Email – bwinjum@oarc.ucla.edu