Introduction to Version Control with Git
Part 1

Ben Winjum
OARC

November 13, 2024

What is version control and why should I use it?

www.phdcomics.com

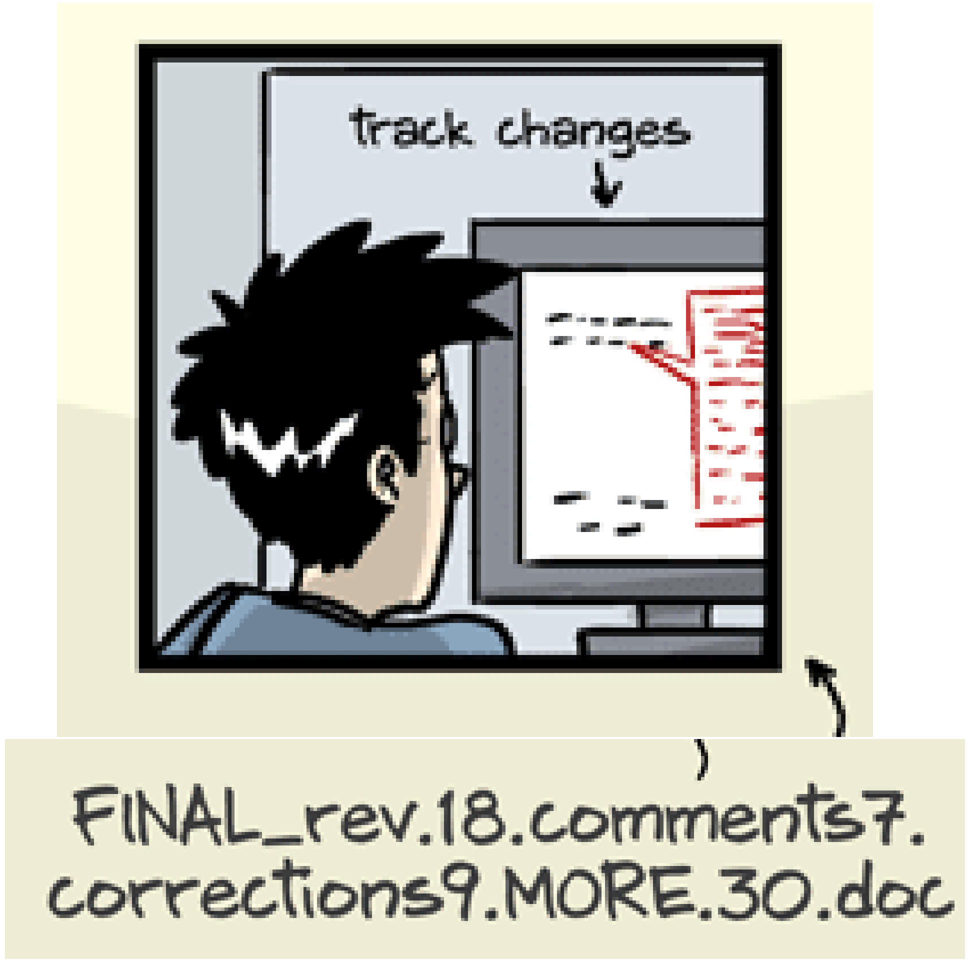
www.phdcomics.com

www.phdcomics.com

www.phdcomics.com

www.phdcomics.com

www.phdcomics.com



www.phdcomics.com

www.phdcomics.com

www.phdcomics.com

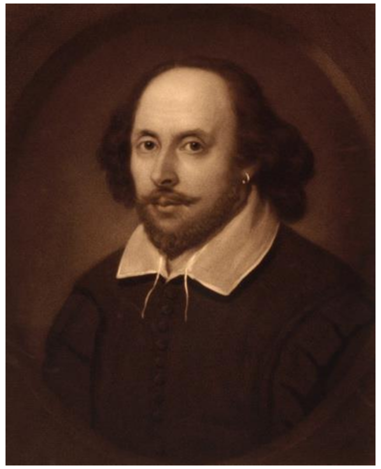
Version Control

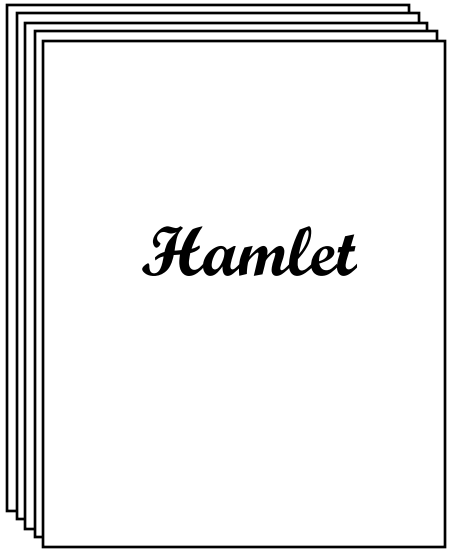
Save the current state (FINAL.doc)

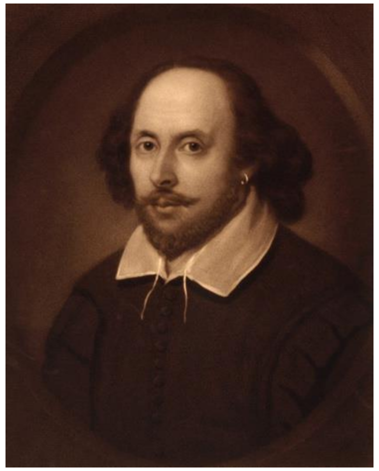
and the entire history of your changes

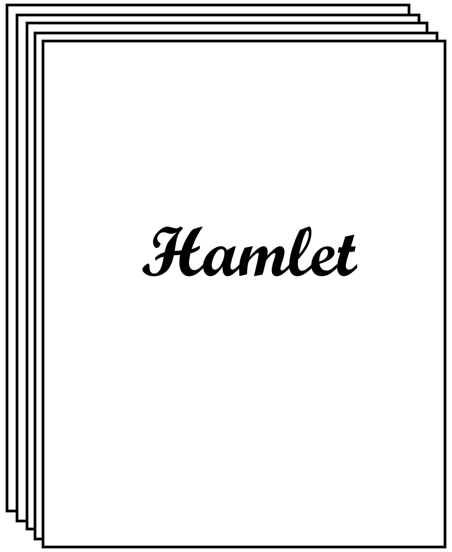
www.phdcomics.com

Example: Shakespeare writing Hamlet



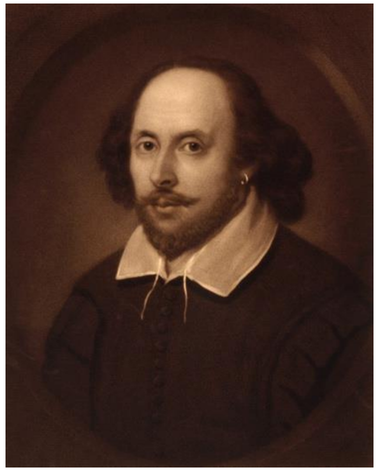


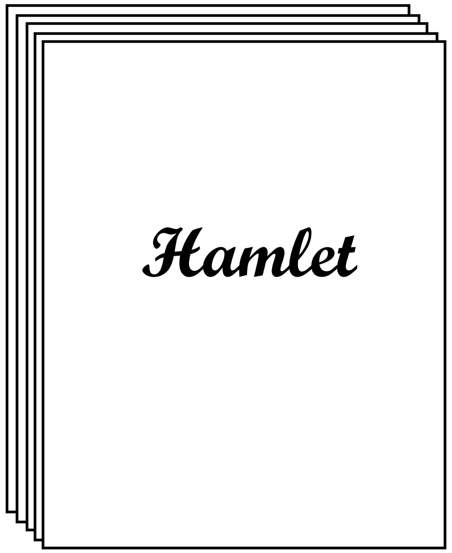




Act III, Scene I

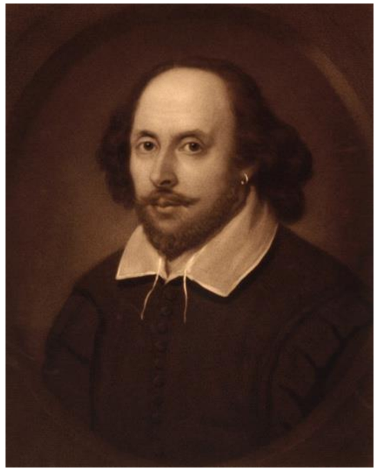
Hamlet: ….

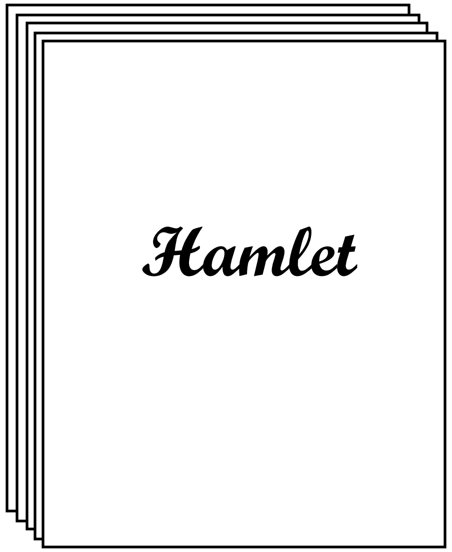




Act III, Scene I

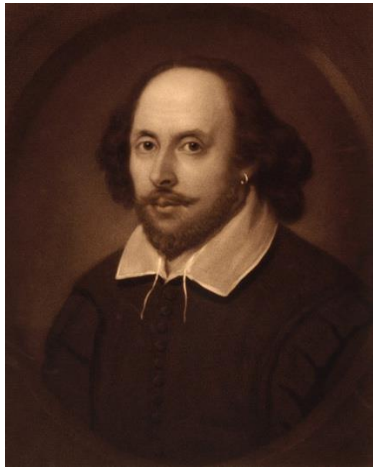
Hamlet:
To be or not to be,
It makes one wonder
About the skies and
The seas, and oysters
And things….

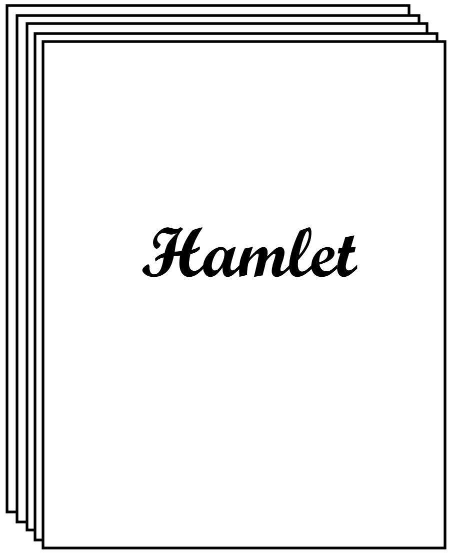




Act III, Scene I

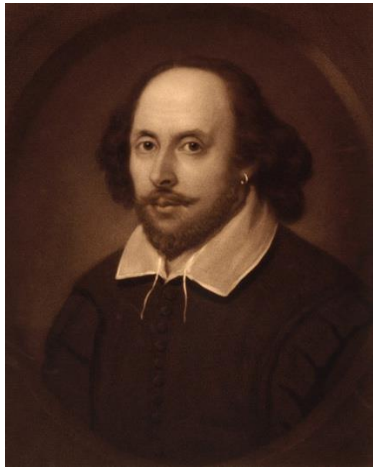
Hamlet:
To be or not to be,
That is the question
It makes one wonder
About the skies and
The seas, and oysters
And things….

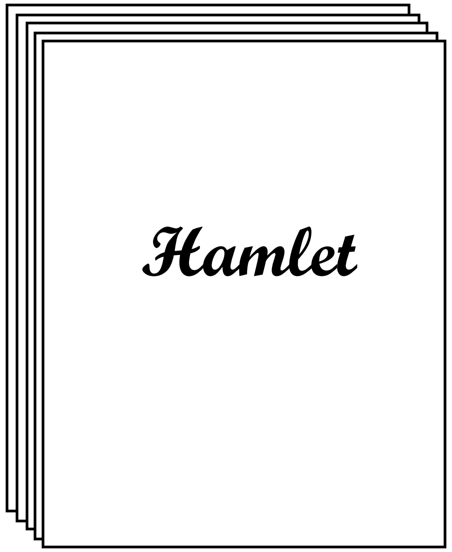




Act III, Scene I

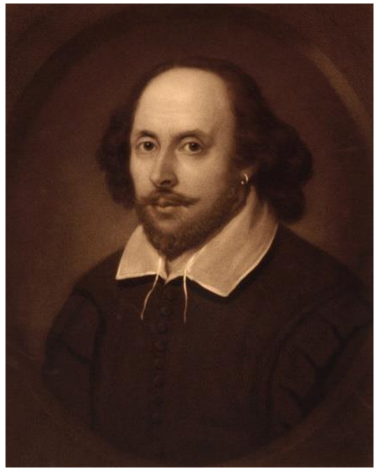
Hamlet:
To be or not to be,
That is the question
~~It~~ That makes one wonder
About the skies and
The seas, and oysters
And things….

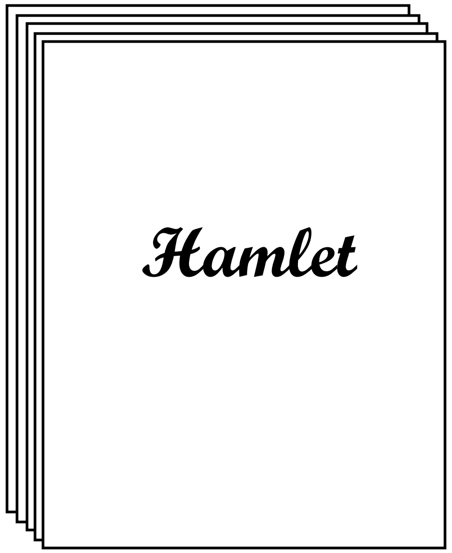




Act III, Scene I

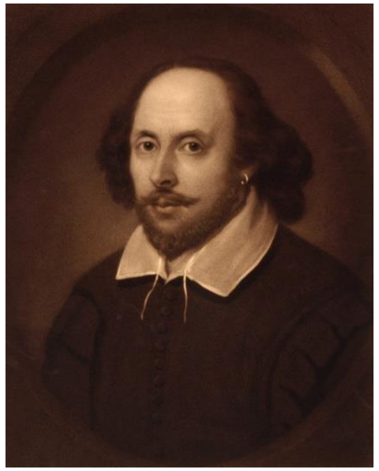
Hamlet:
To be or not to be,
That is the question
~~It~~ That makes one wonder
About the skies and
The seas of outrageous fortunes
~~, and oysters~~
~~And things~~....

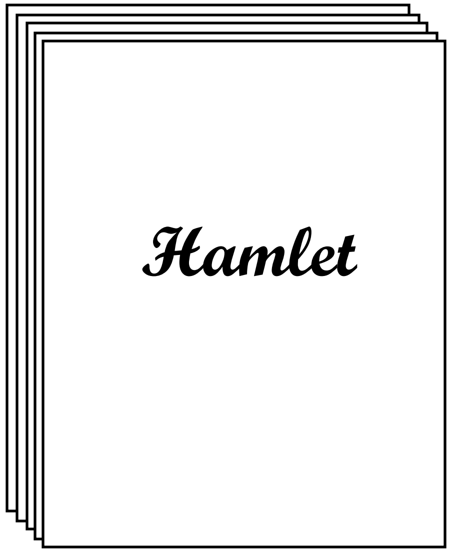




Act III, Scene I

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



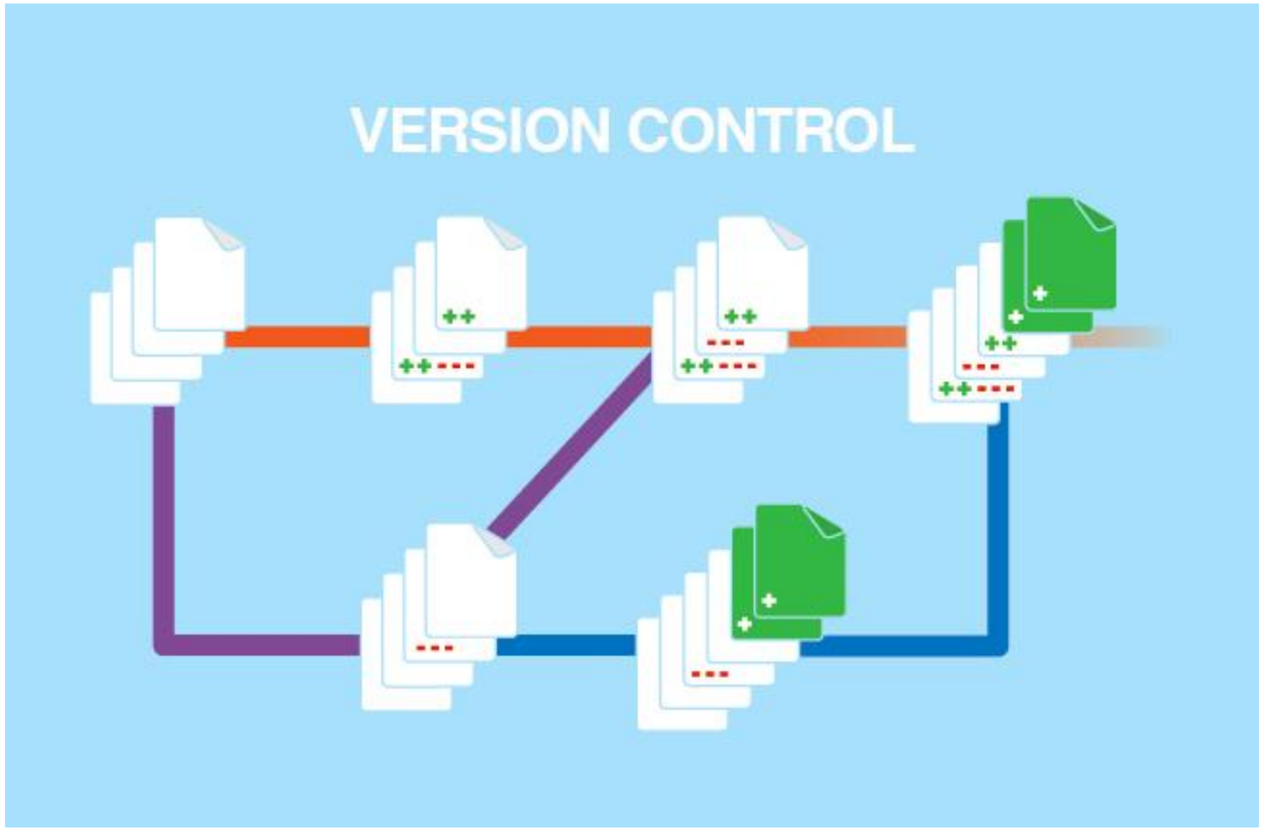


Act III, Scene I

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

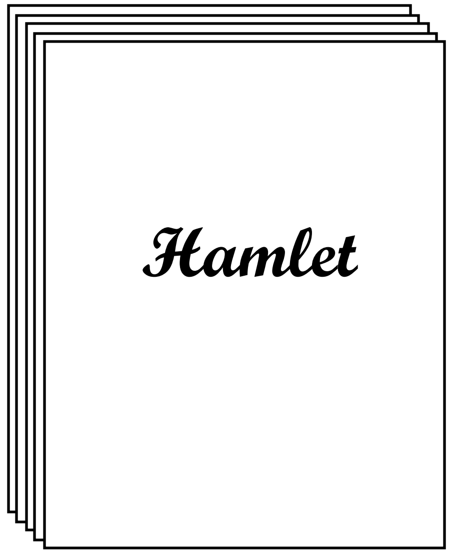
If only Shakespeare had been able to use git….

Git is a **Version Control System**

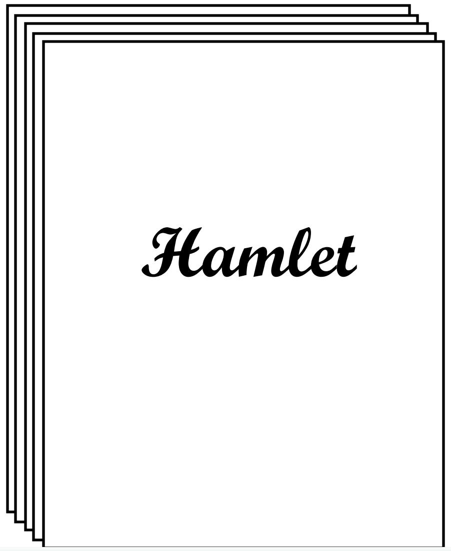


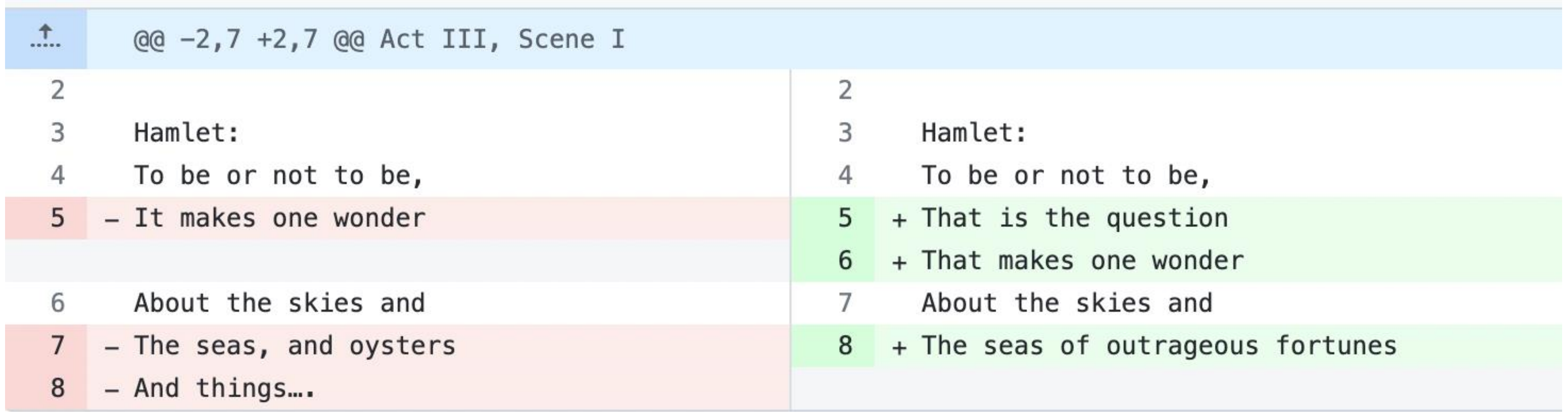
https://www.linode.com/docs/development/version-control/introduction-to-version-control/1204-image_versioning_intro_1.jpg

Git repository:
the collection of files



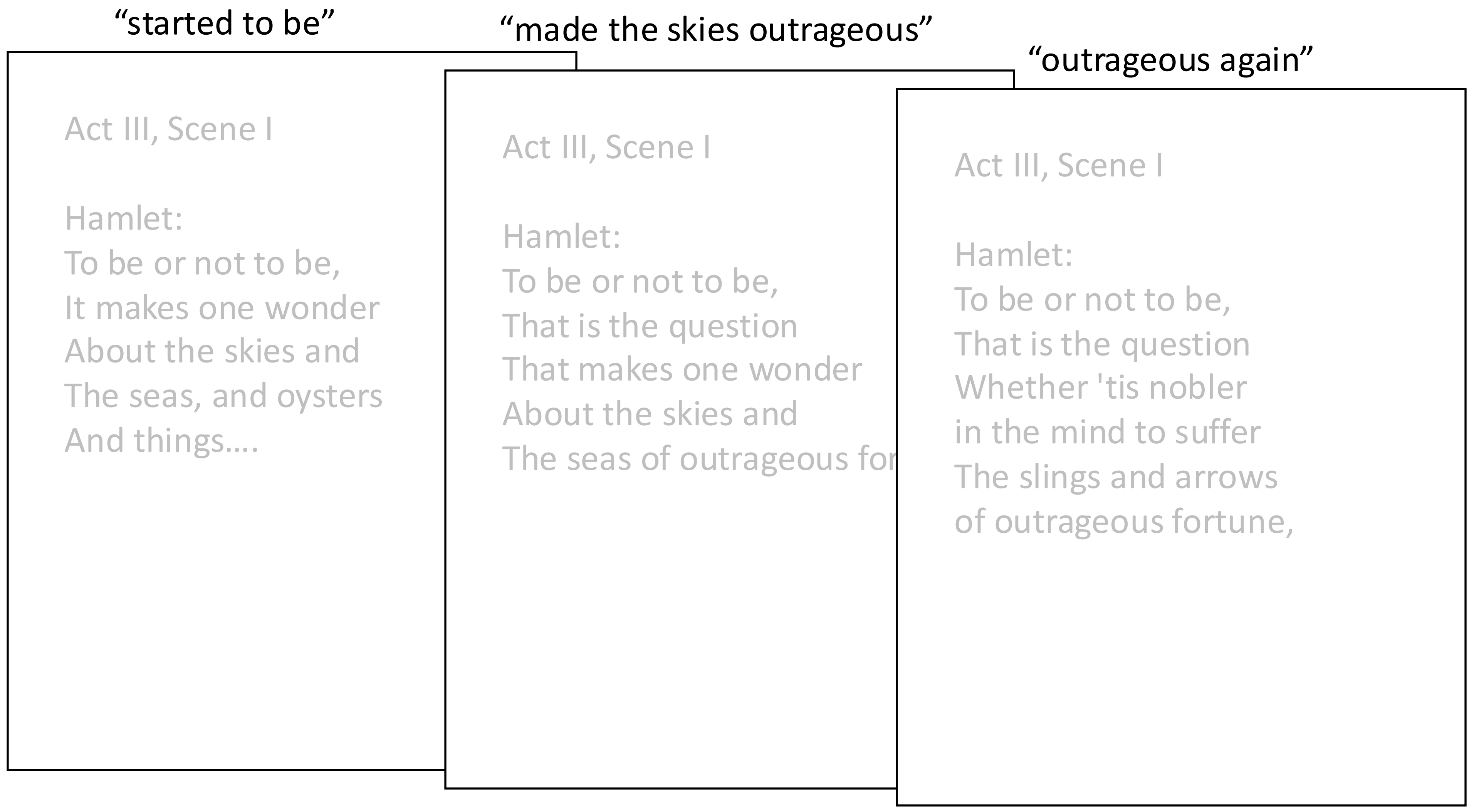
Git repository:
the collection of files
and
the change history





```
@@ -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



“started to be”

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….

“made the skies outrageous”

Act III, Scene I

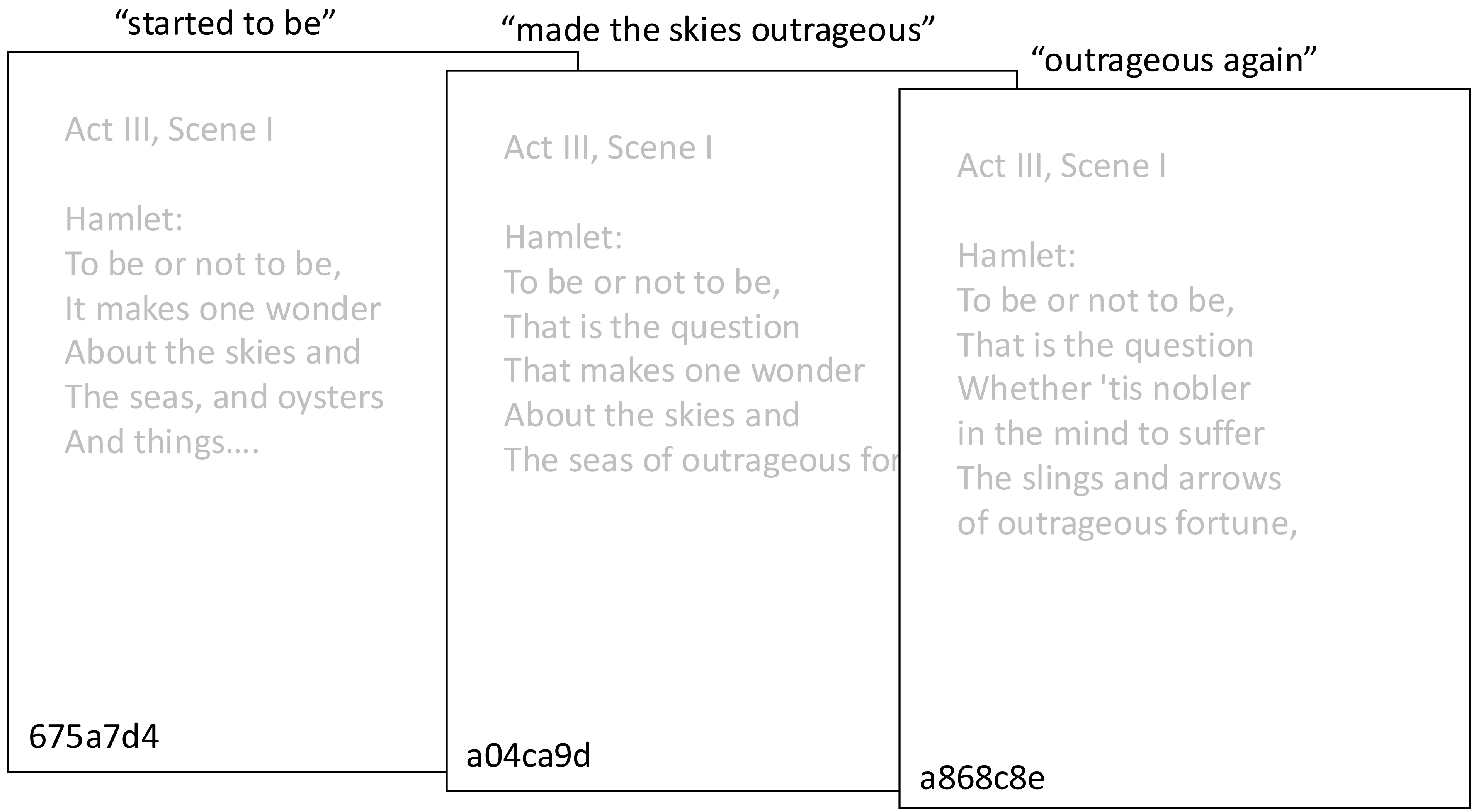
Hamlet:
To be or not to be,
That is the question
That makes one wonder
About the skies and
The seas of outrageous for

“outrageous again”

Act III, Scene I

Hamlet:
To be or not to be,
That is the question
Whether 'tis nobler
in the mind to suffer
The slings and arrows
of outrageous fortune,

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



“started to be”

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….

675a7d4

“made the skies outrageous”

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 for

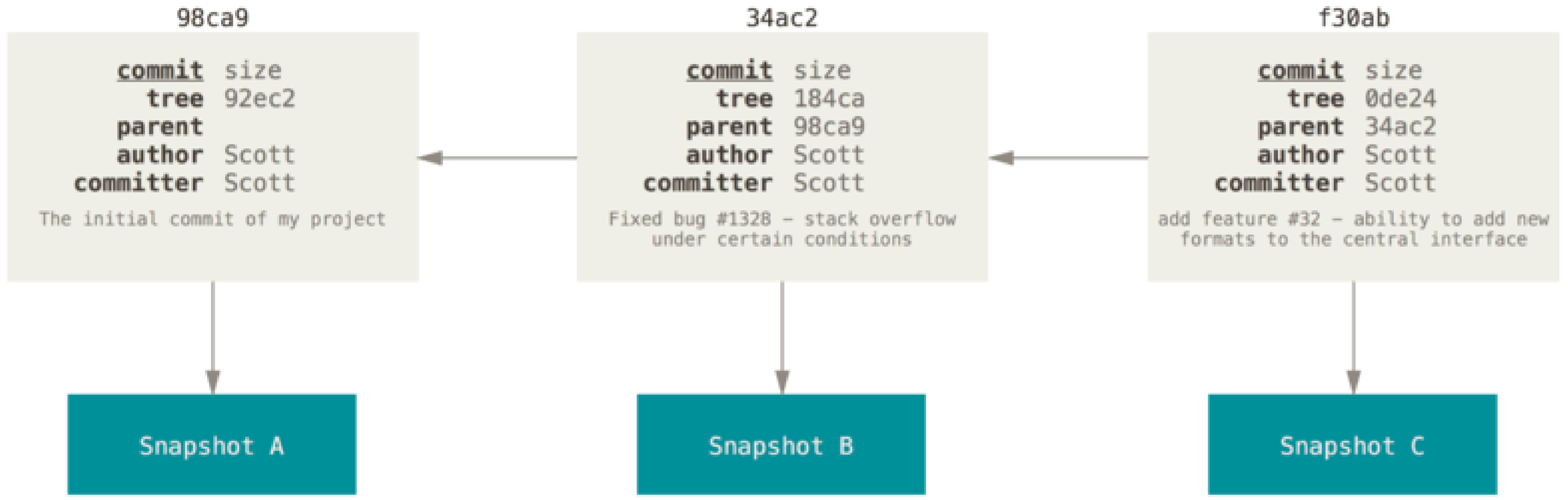
a04ca9d

“outrageous again”

Act III, Scene I

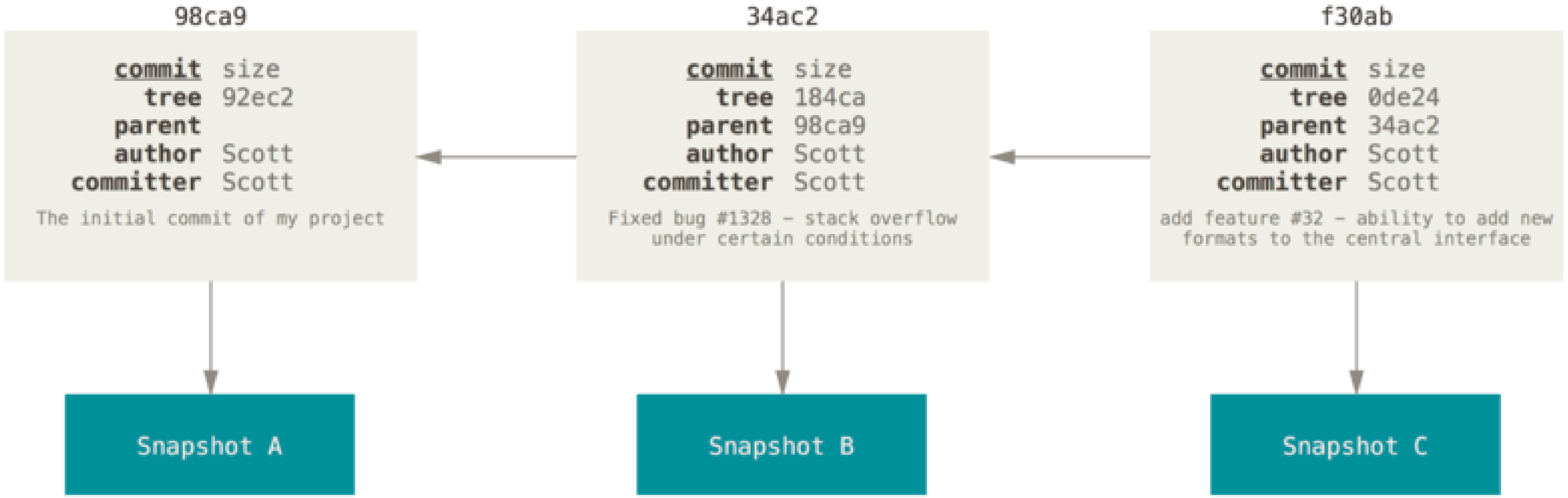
Hamlet:
To be or not to be,
That is the question
Whether 'tis nobler
in the mind to suffer
The slings and arrows
of outrageous fortune,

a868c8e



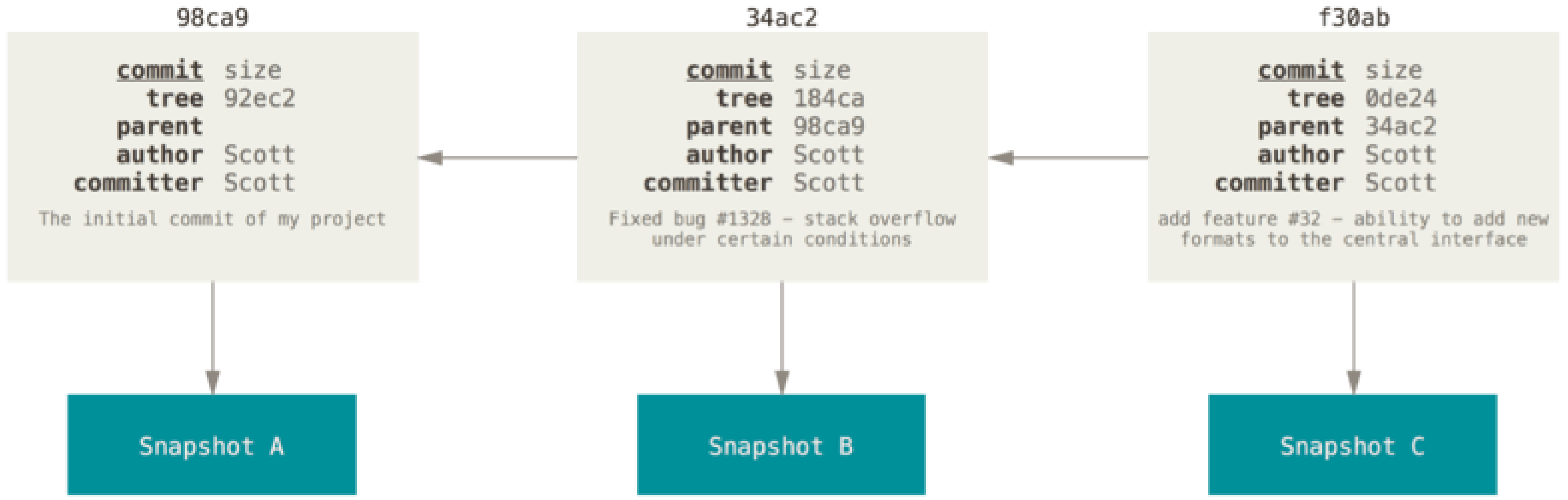
Images taken from the Pro Git book -- freely available online and recommended for further reading
https://git-scm.com/book/en/v2

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

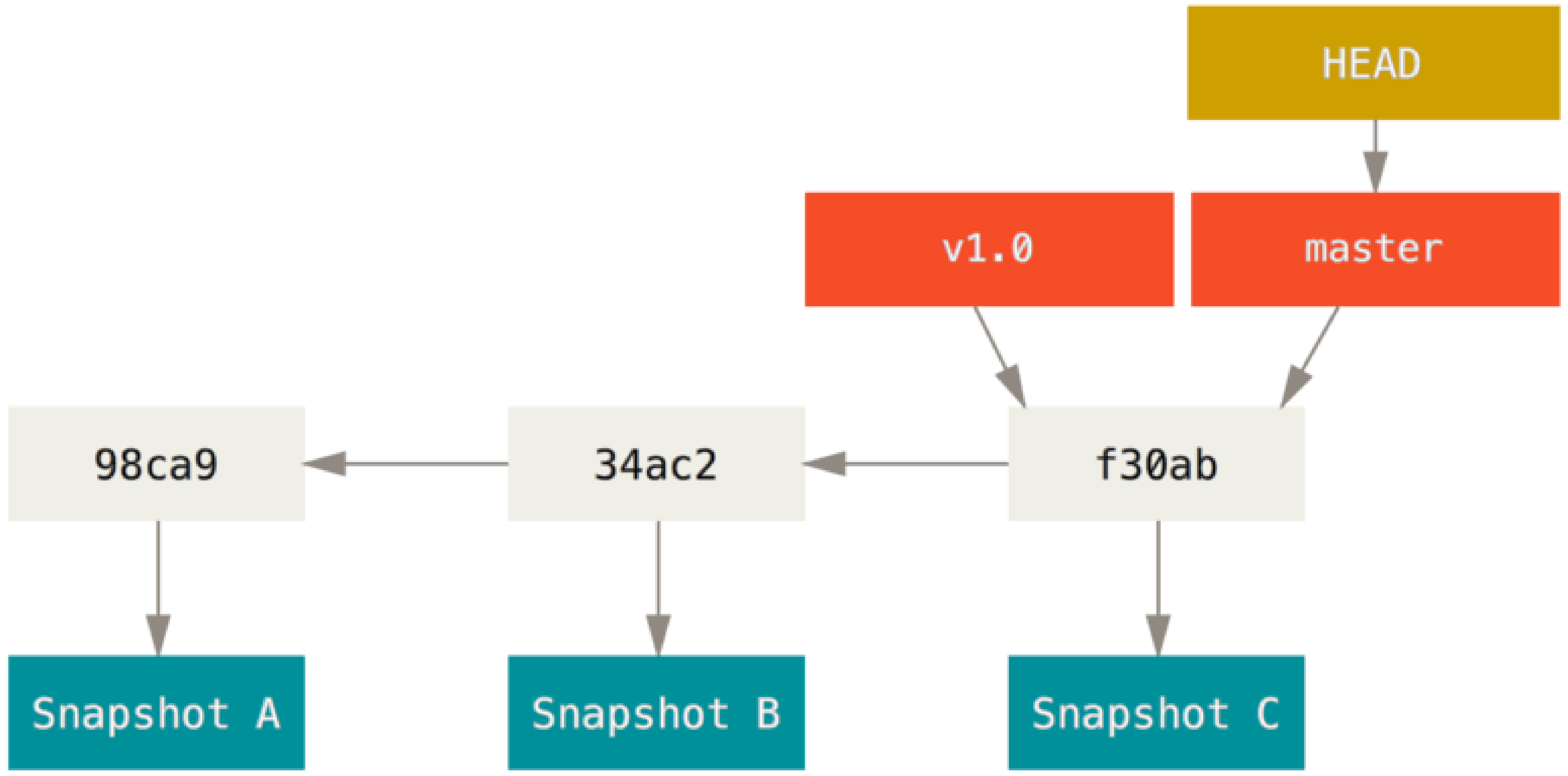


Images taken from the Pro Git book -- freely available online and recommended for further reading
https://git-scm.com/book/en/v2

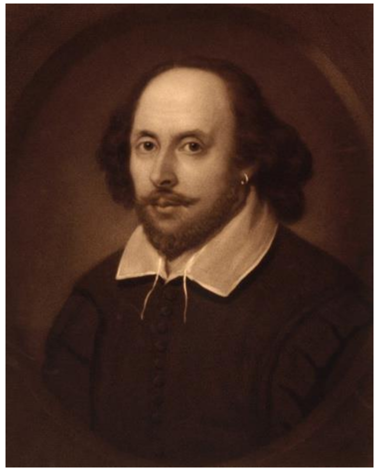
But wait…. Shakespeare isn't the author here….
Best practice: don't commit as an unrecognized author

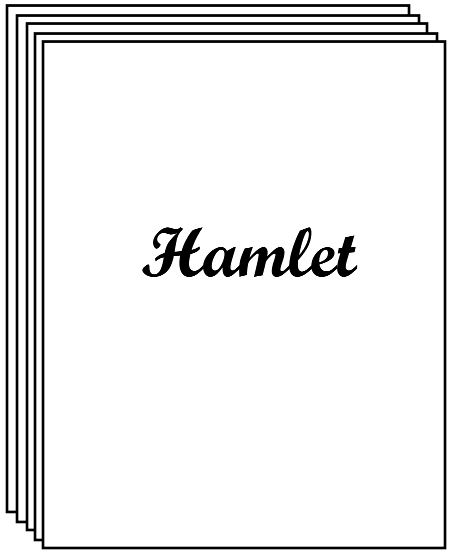


Images taken from the Pro Git book -- freely available online and recommended for further reading
https://git-scm.com/book/en/v2

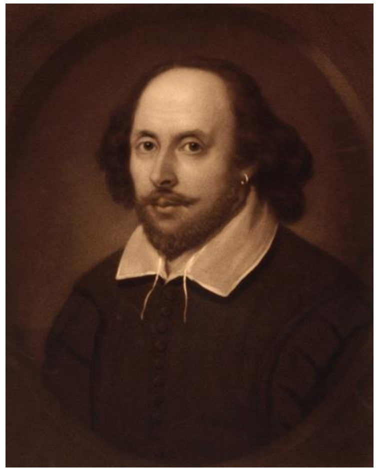


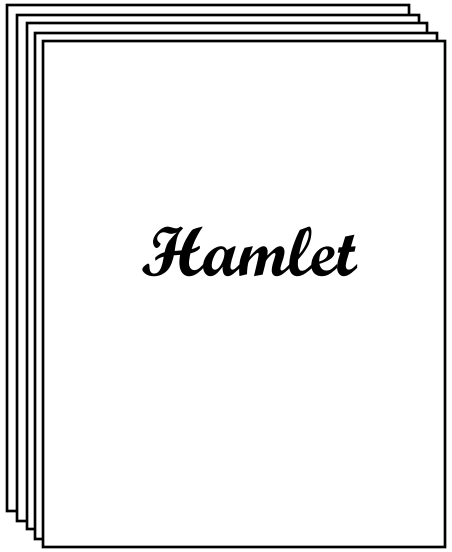
Images taken from the Pro Git book -- freely available online and recommended for further reading
https://git-scm.com/book/en/v2



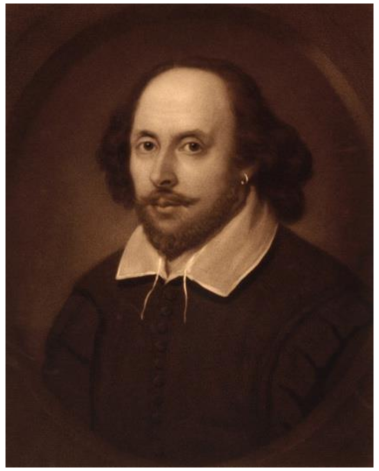


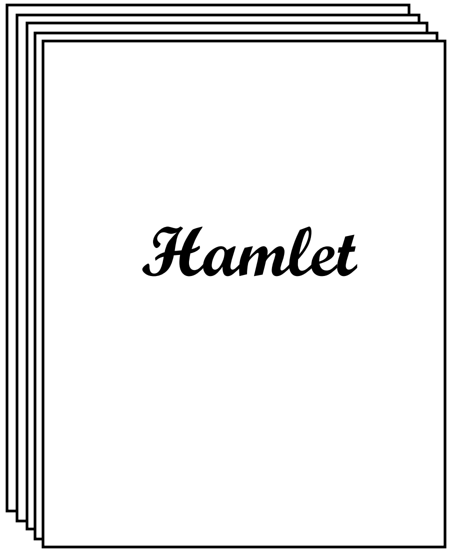
If only Shakespeare had been able to use git….





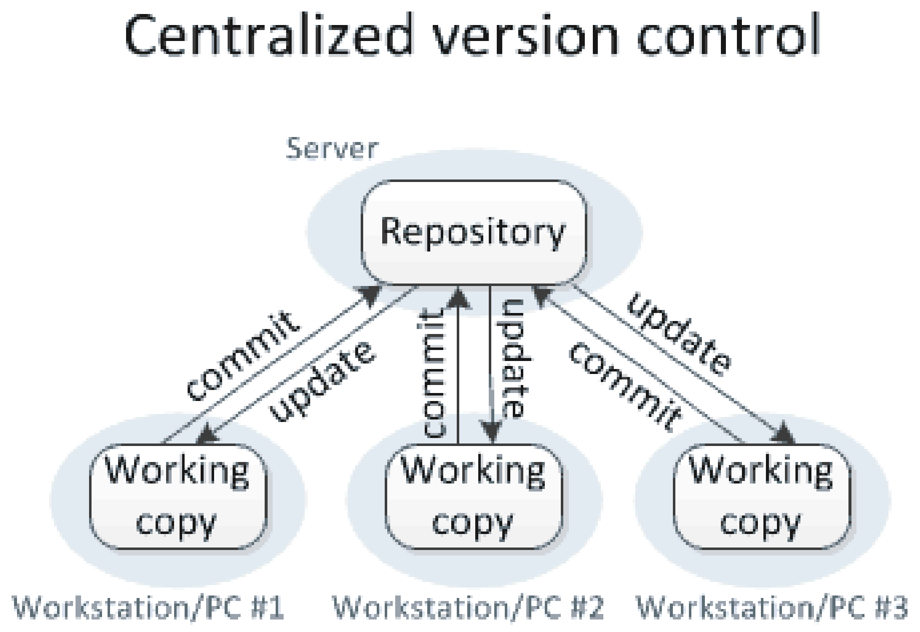
If only Shakespeare had been able to use git….

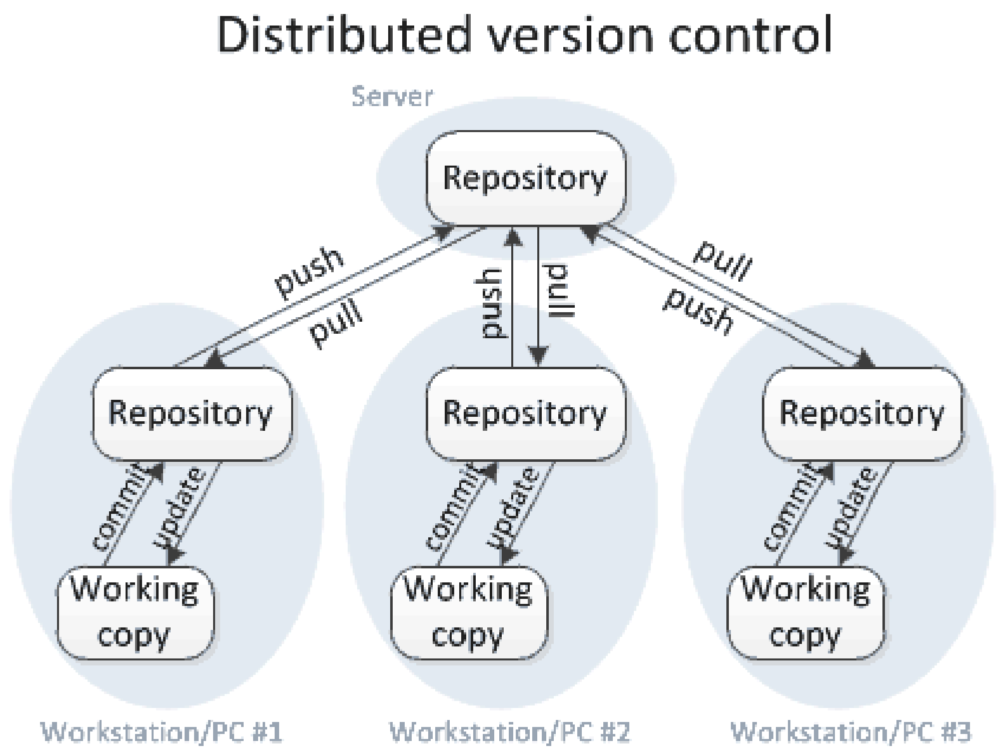




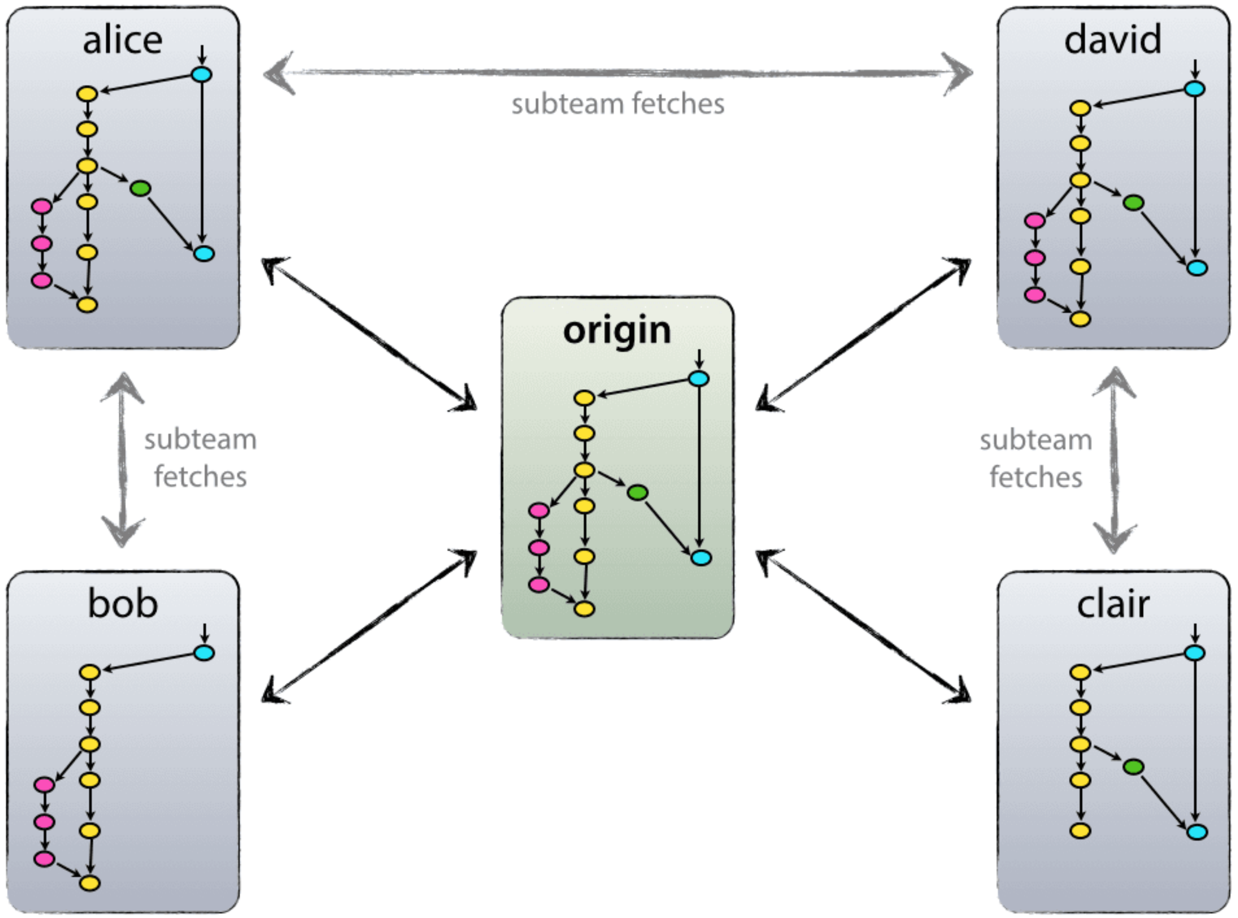
git is used for *collaboration!*

Git is a **Distributed** Version Control System

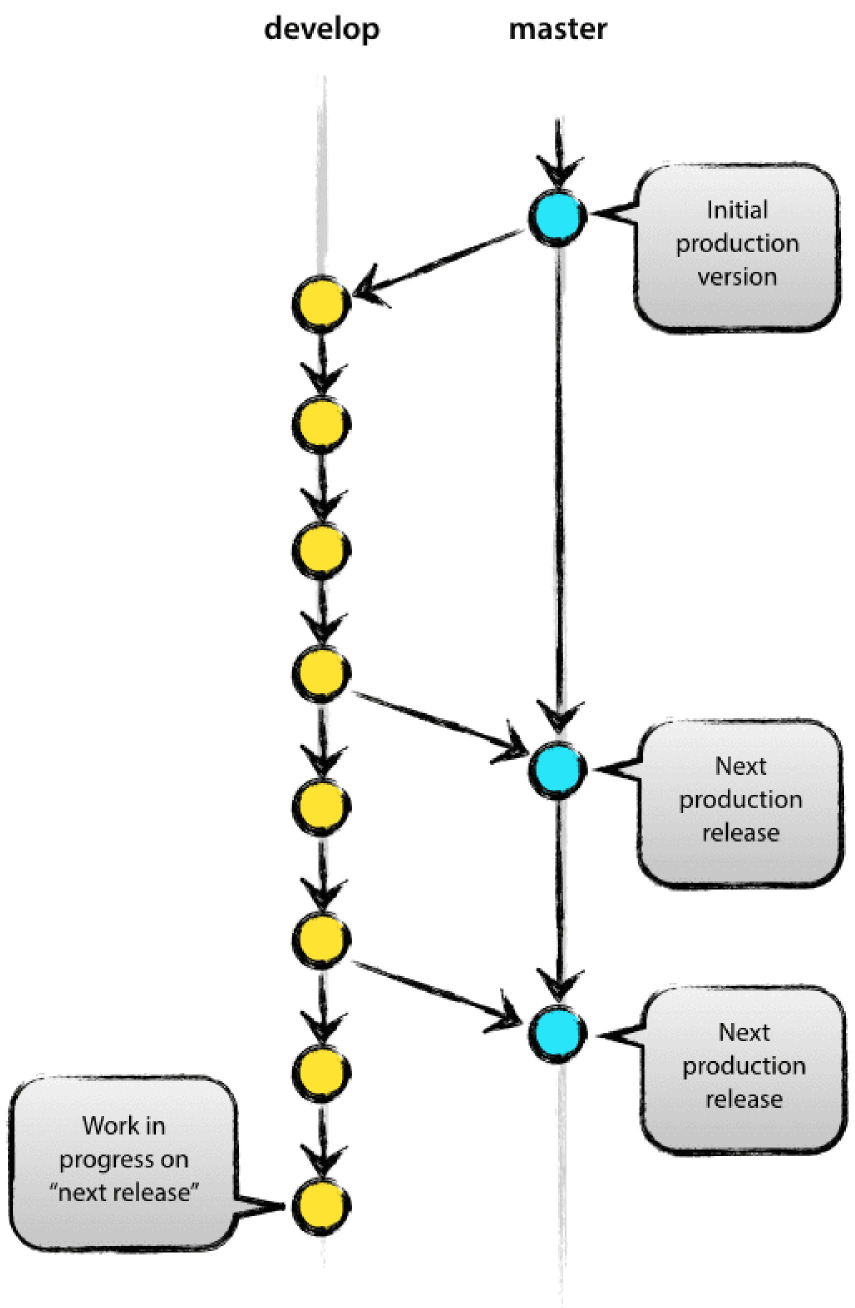




Git is a **Distributed** Version Control System



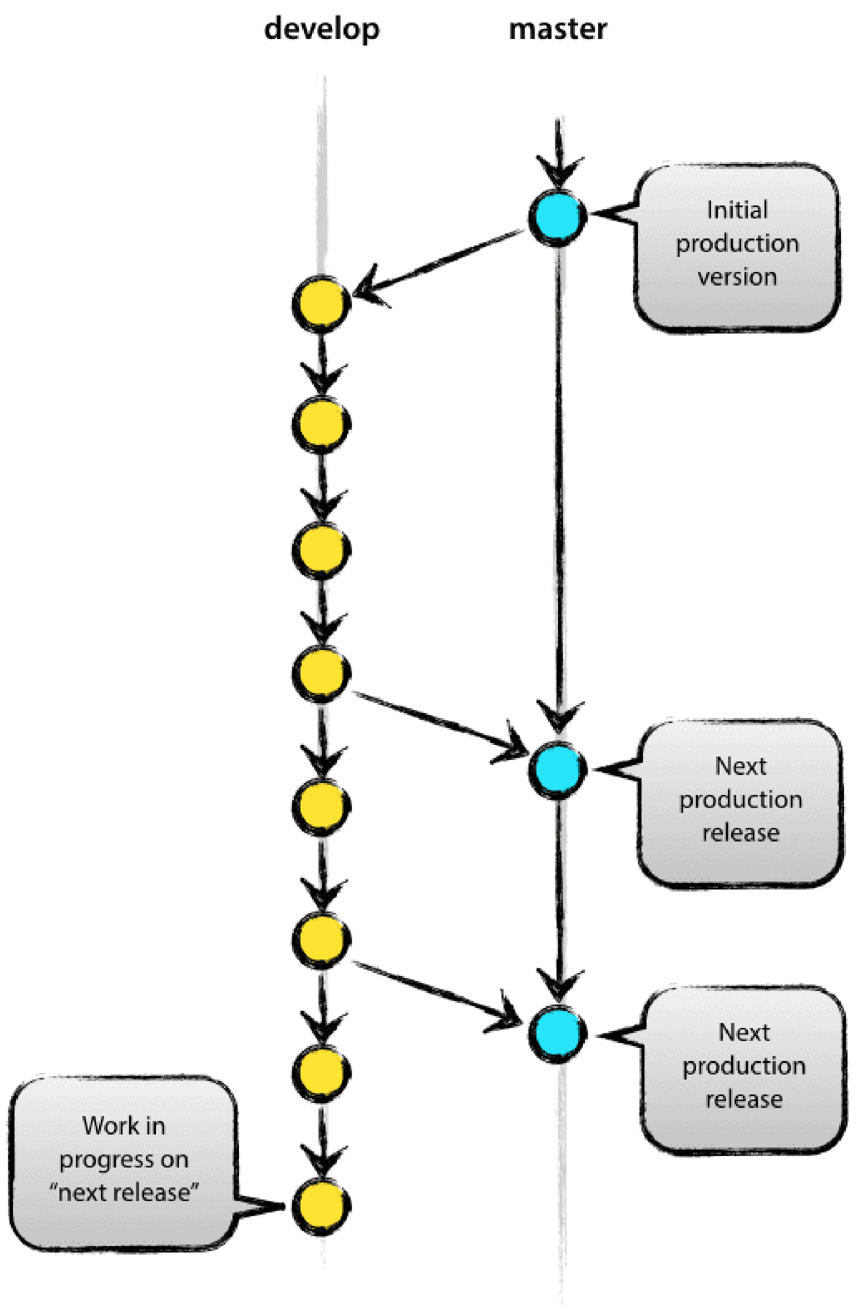
https://nvie.com/posts/a-successful-git-branching-model/


develop
master
Initial production version
Next production release
Next production release
Work in progress on "next release"

Development with git can occur in ”branches”

These are separate but related lines of modifications

https://nvie.com/posts/a-successful-git-branching-model/


develop
master
Initial production version
Next production release
Next production release
Work in progress on "next release"

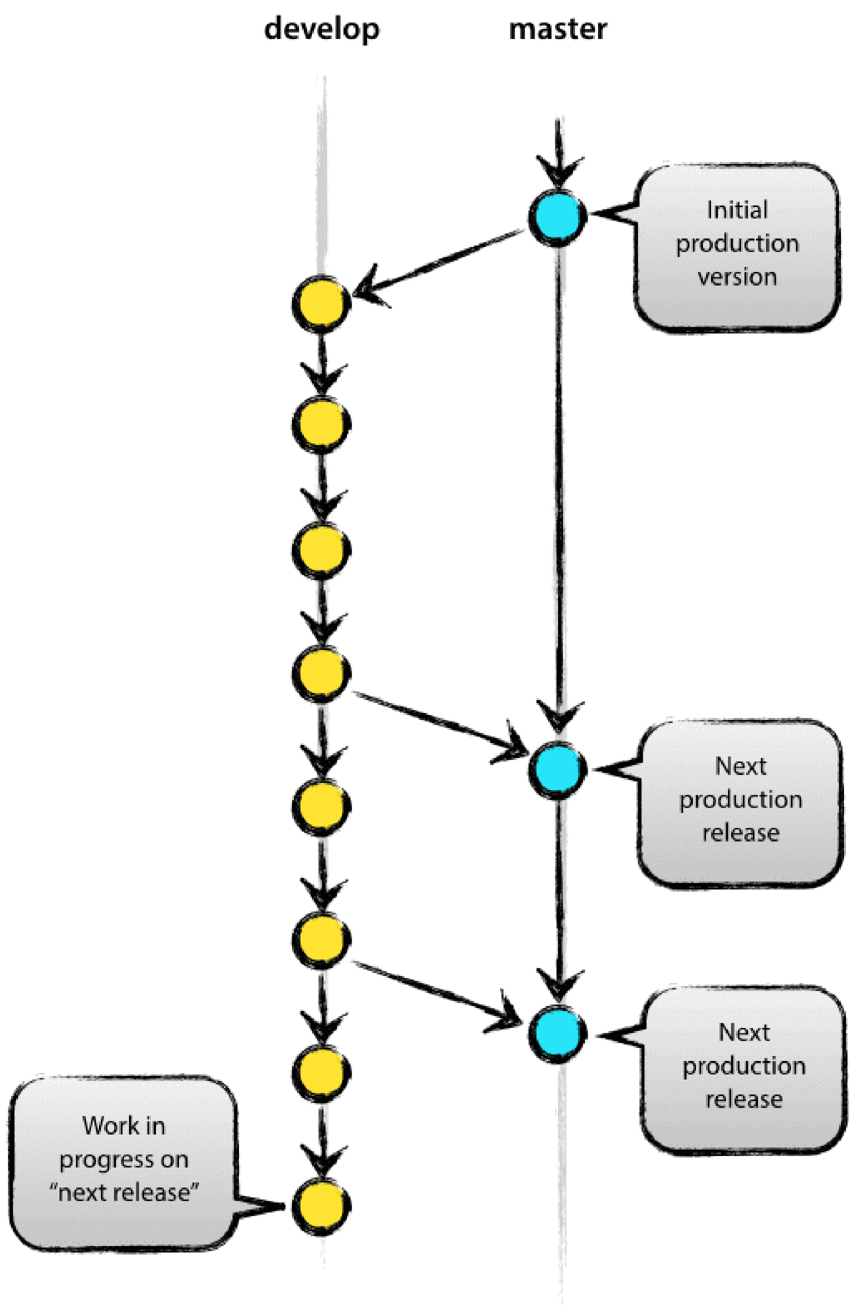
Development with git can occur in ”branches”

These are separate but related lines of modifications

Example:

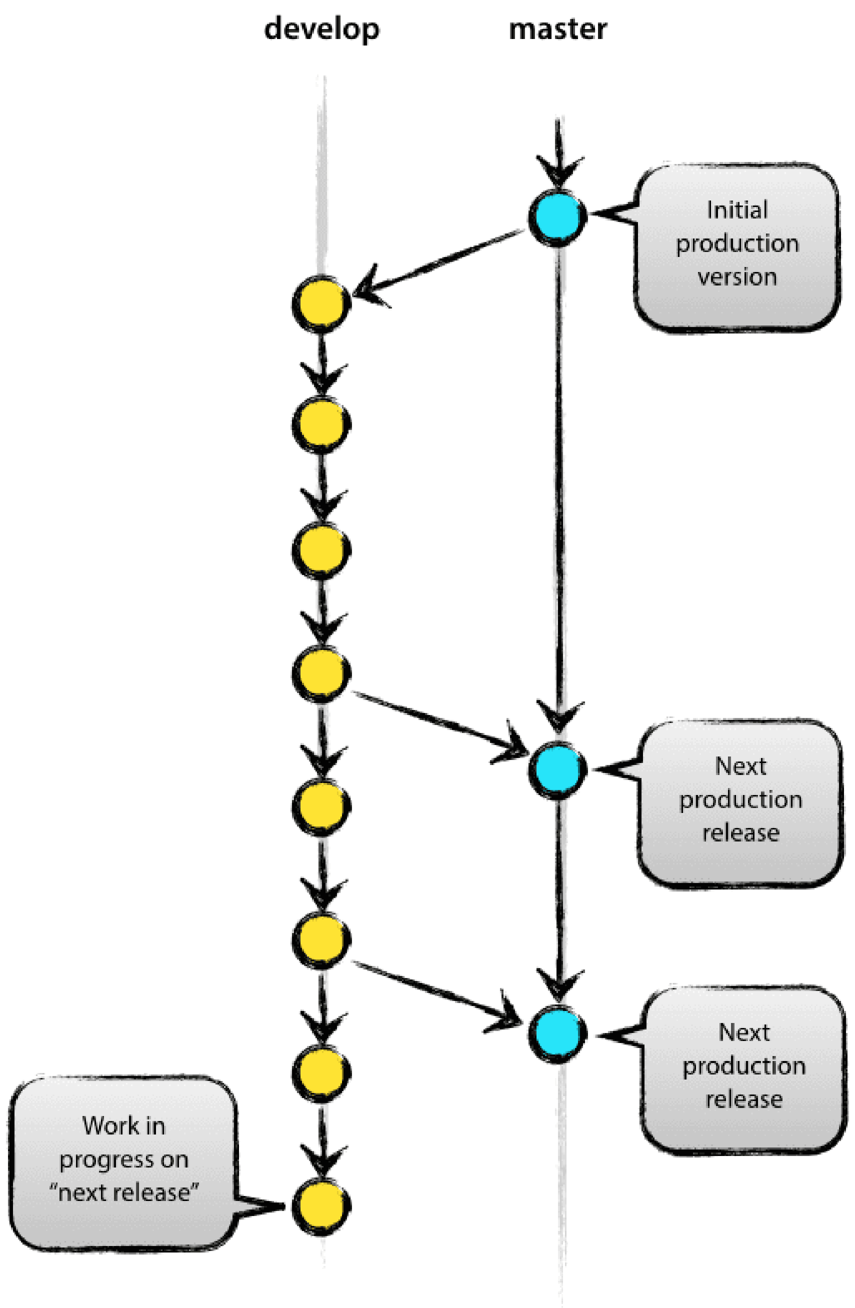
- Develop a new feature on a feature branch
- When done, “merge” the feature branch with the main branch

https://nvie.com/posts/a-successful-git-branching-model/


develop
master
Initial production version
Next production release
Next production release
Work in progress on "next release"

Say a collaborator works on a new feature

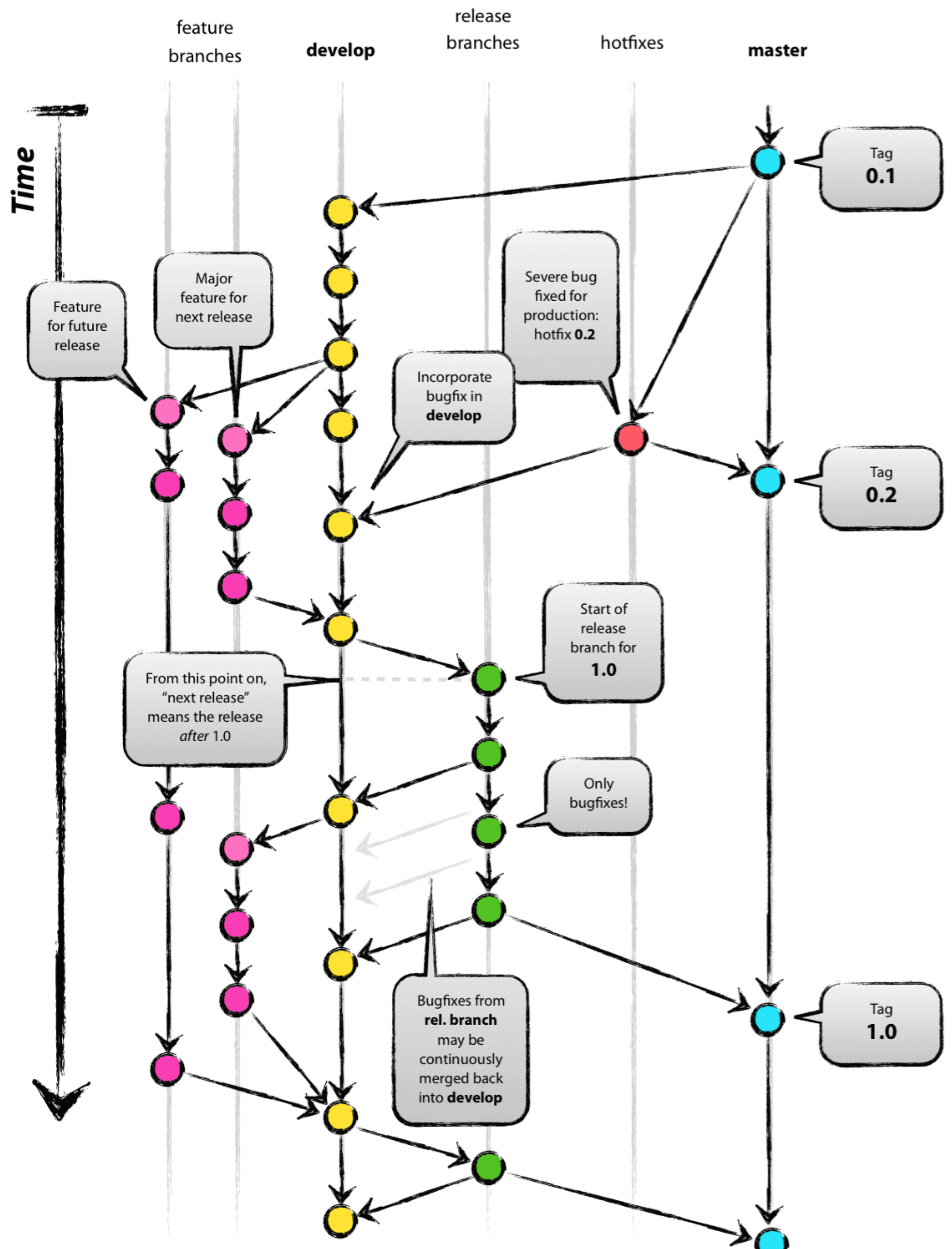
https://nvie.com/posts/a-successful-git-branching-model/


develop
master
Initial production version
Next production release
Next production release
Work in progress on "next release"

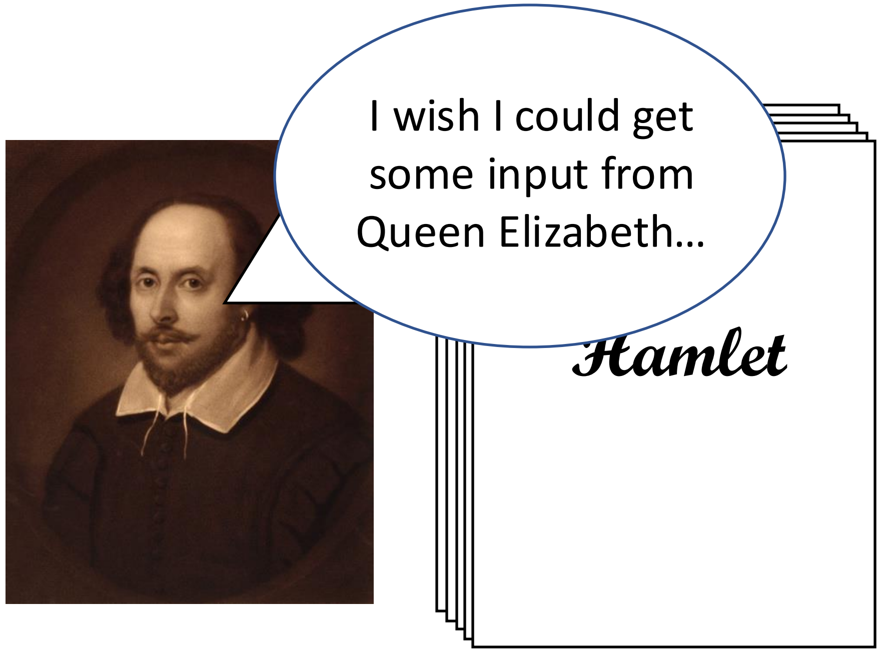
Say a collaborator works on a new feature

When they want to merge it with the main branch
-> "Hey, repo-owner, merge this into the main branch"
-> pull request

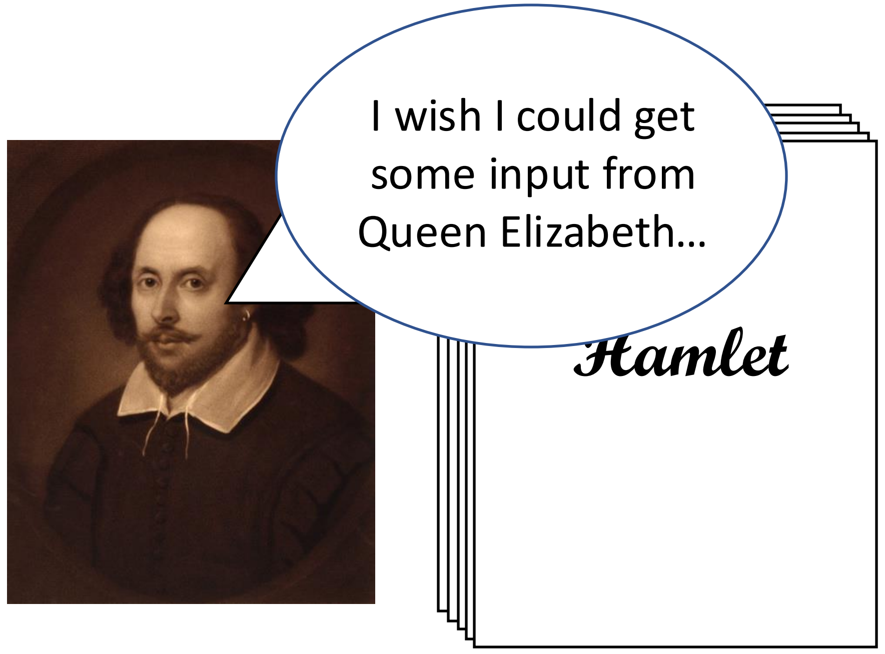
https://nvie.com/posts/a-successful-git-branching-model/


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix **0.2**
Incorporate bugfix in **develop**
Tag 0.2
Start of release branch for **1.0**
From this point on, "next release" means the release *after* 1.0
Only bugfixes!
Bugfixes from **rel. branch** may be continuously merged back into **develop**
Tag 1.0

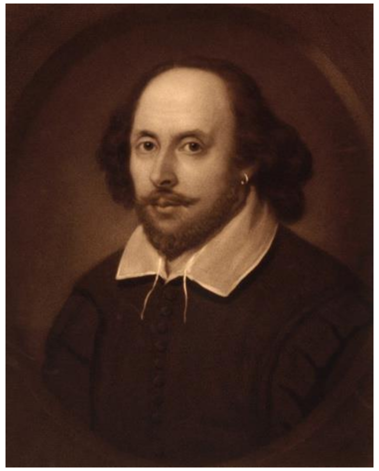
https://nvie.com/posts/a-successful-git-branching-mo

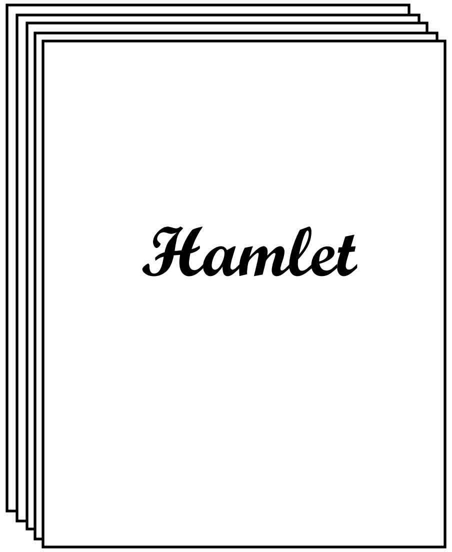


git is used for *collaboration!*

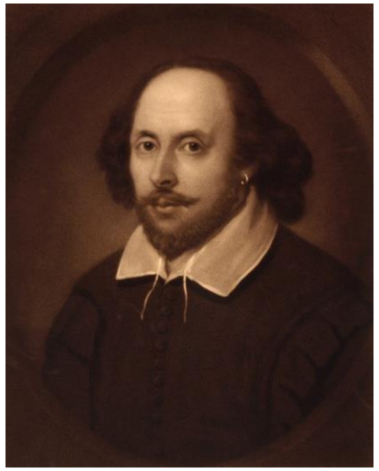


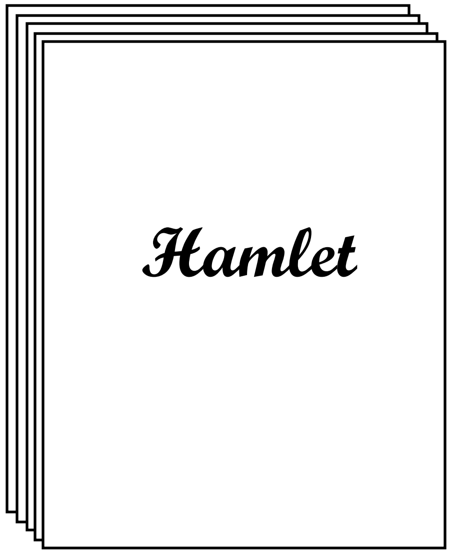
GitHub (and GitLab and Bitbucket) are git-based platforms that combine git with a rich set of tools for development and collaboration



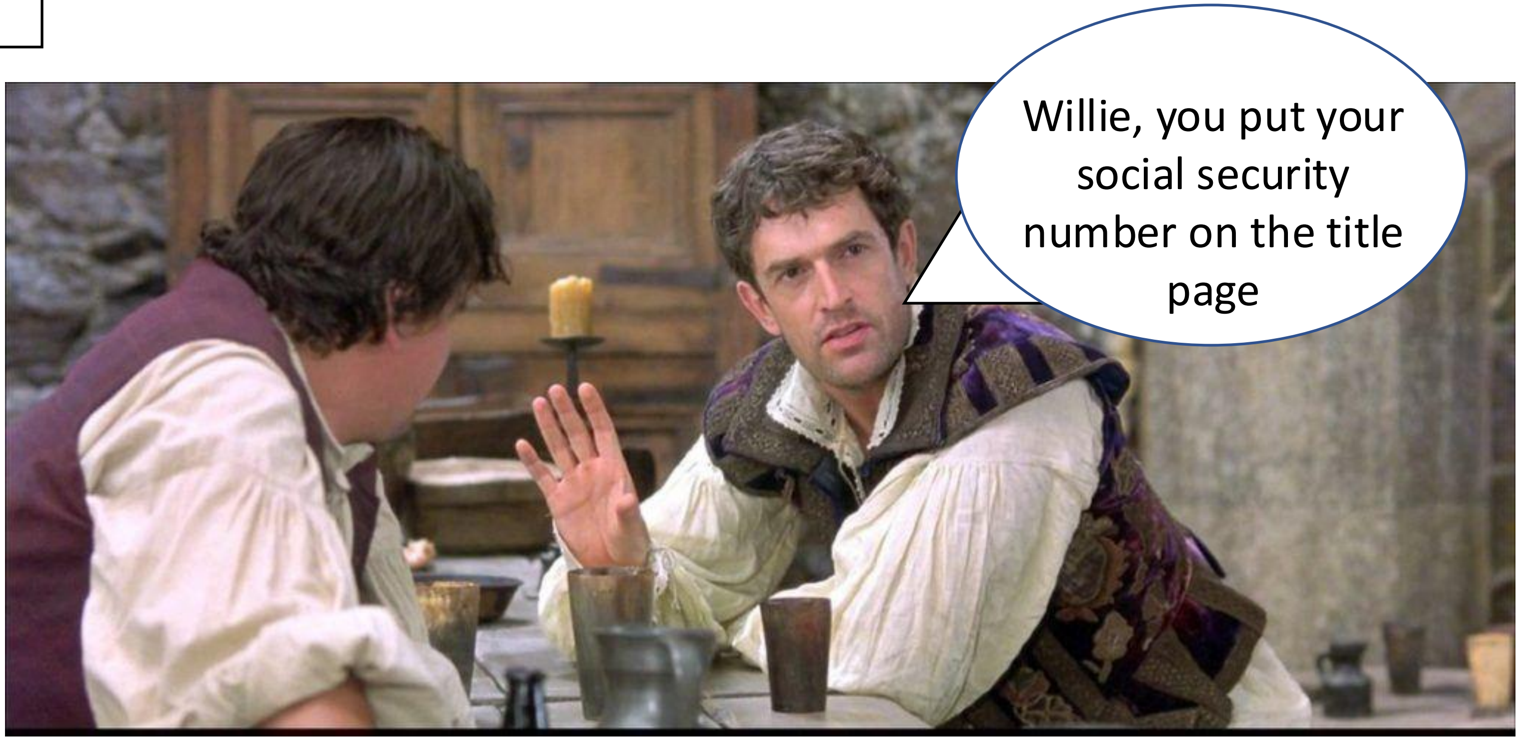

Hamlet

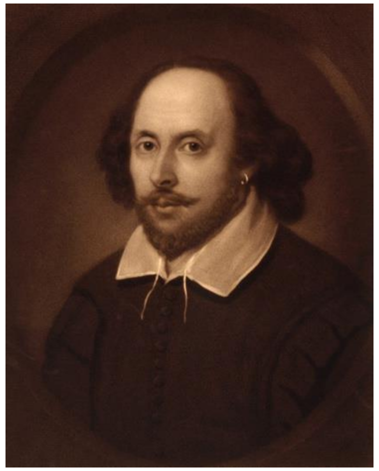
Willie, you put your social security number on the title page

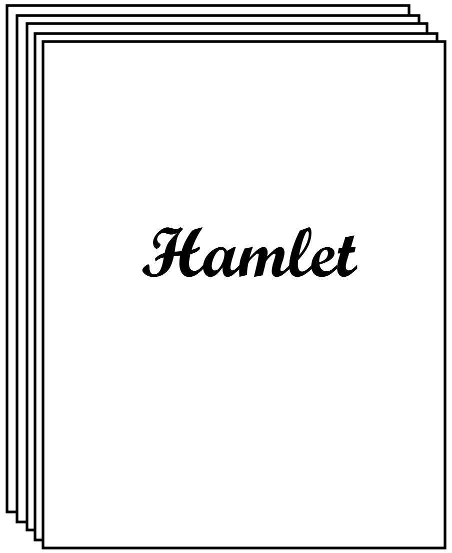




github.com is a publicly accessible platform


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



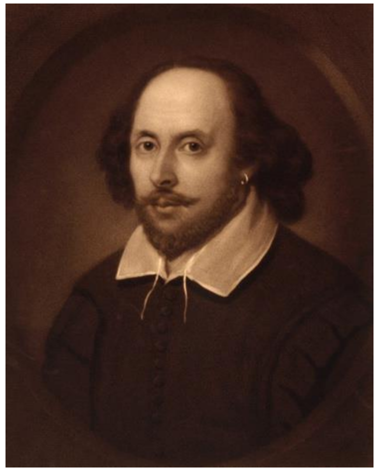


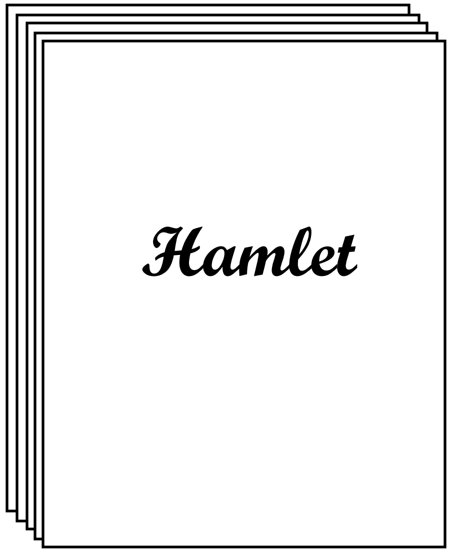
Best practices:

- 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

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





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

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

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

- I am going to be demo-ing git inside of JupyterHub
  - This is like a virtual machine with Linux
  - Let's go there now and clone the repo with a git

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
  - `export PS1='\w $ '`
    - Simplifies the command prompt

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
```

HELP!!

Always remember that if you forget a Git command, you can access the list of common commands with --help:

```
$ git --help
```

For Git commands, you can see command options by using -h and access the Git manual by using --help :

```
$ git commandname -h
$ git commandname --help
 - or -
$ git help commandname
```

A few git commands

- `git config`
  - `See previous slide`
- `git clone repository_name`
  - Get a copy of the repo (e.g. https://github.com/benjum/git-workshop)
- We'll also study:
  - `git init`
  - `git add`
  - `git commit`
  - `git push`

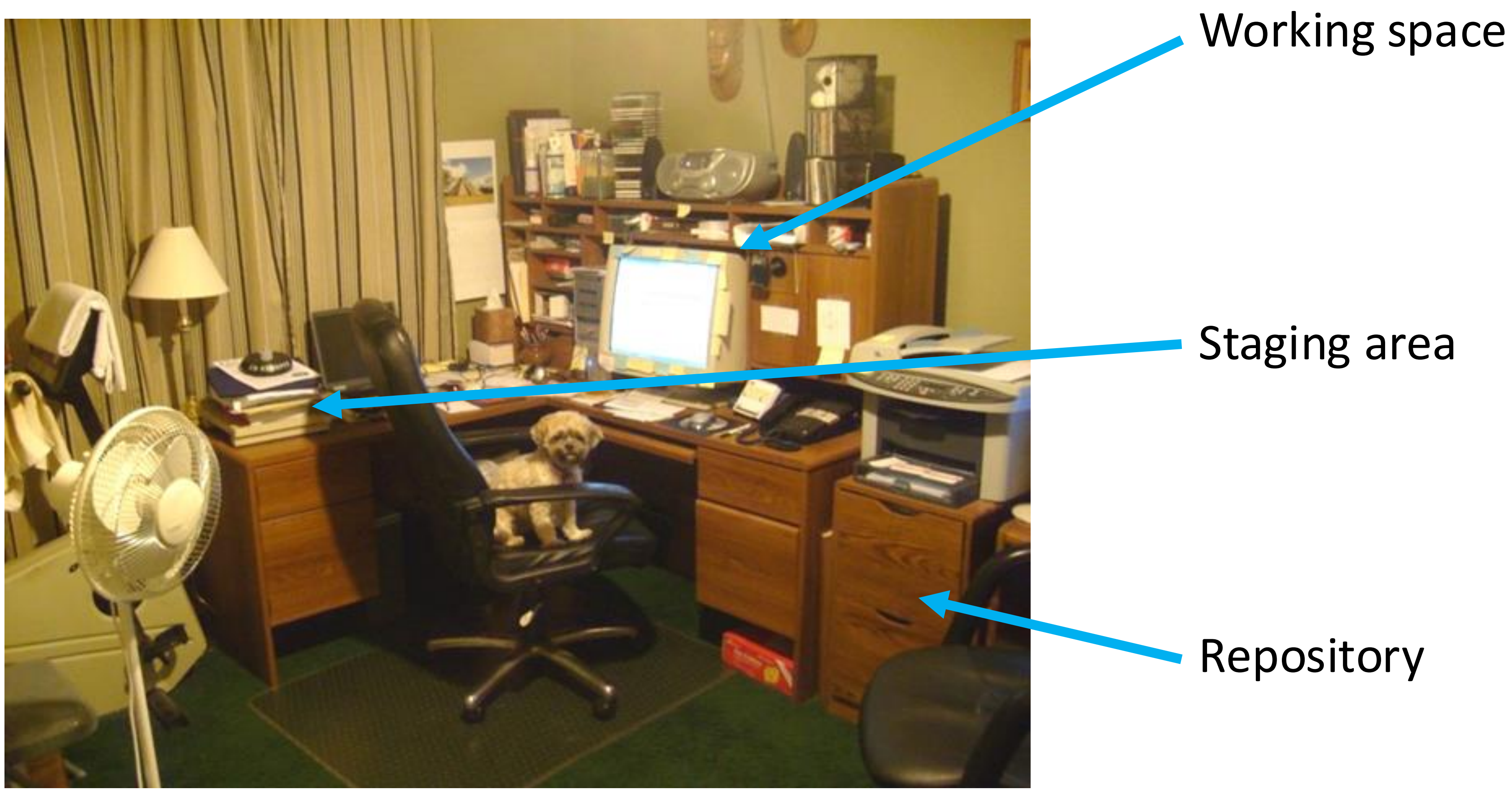
Making a repository

- `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 branch -m <name>`
  - "master" is the default branch name but now "main" is often used
- .git -- Git will store information about the repository in a hidden directory

Adding files and keeping track of versions

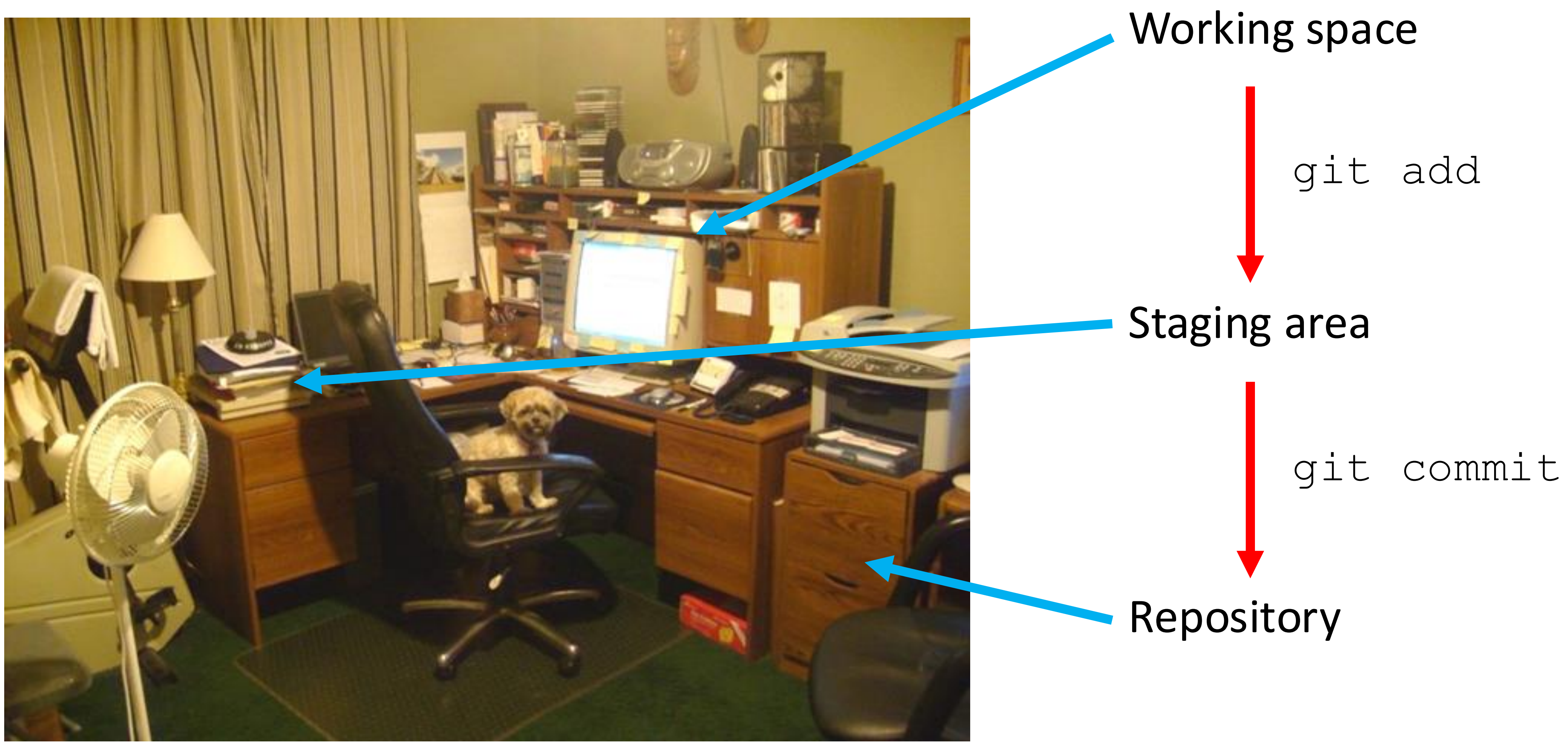
- Recording changes
- Checking status
- Making notes of changes
- Ignoring certain changes

Git is like a desk



https://successfulsoftware.files.wordpress.com/2009/07/officedog2.jpg

Git is like a desk



https://successfulsoftware.files.wordpress.com/2009/07/officedog2.jpg

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

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

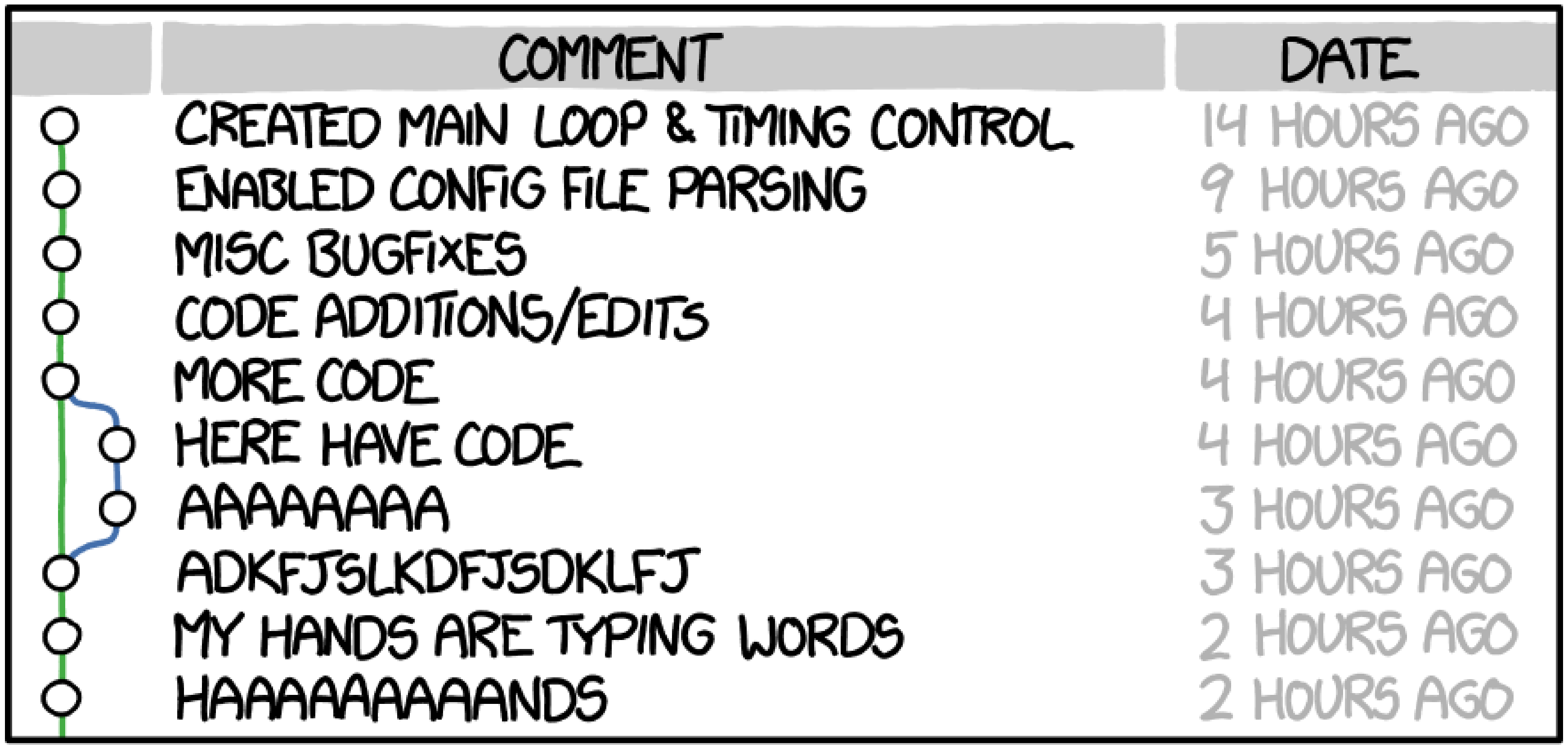
Suppose you use Git to manage a file named `myfile.txt`. After editing the file, which command(s) below would save the changes to the repository?

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

Best Practice: make good commit messages



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	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://tbaggery.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

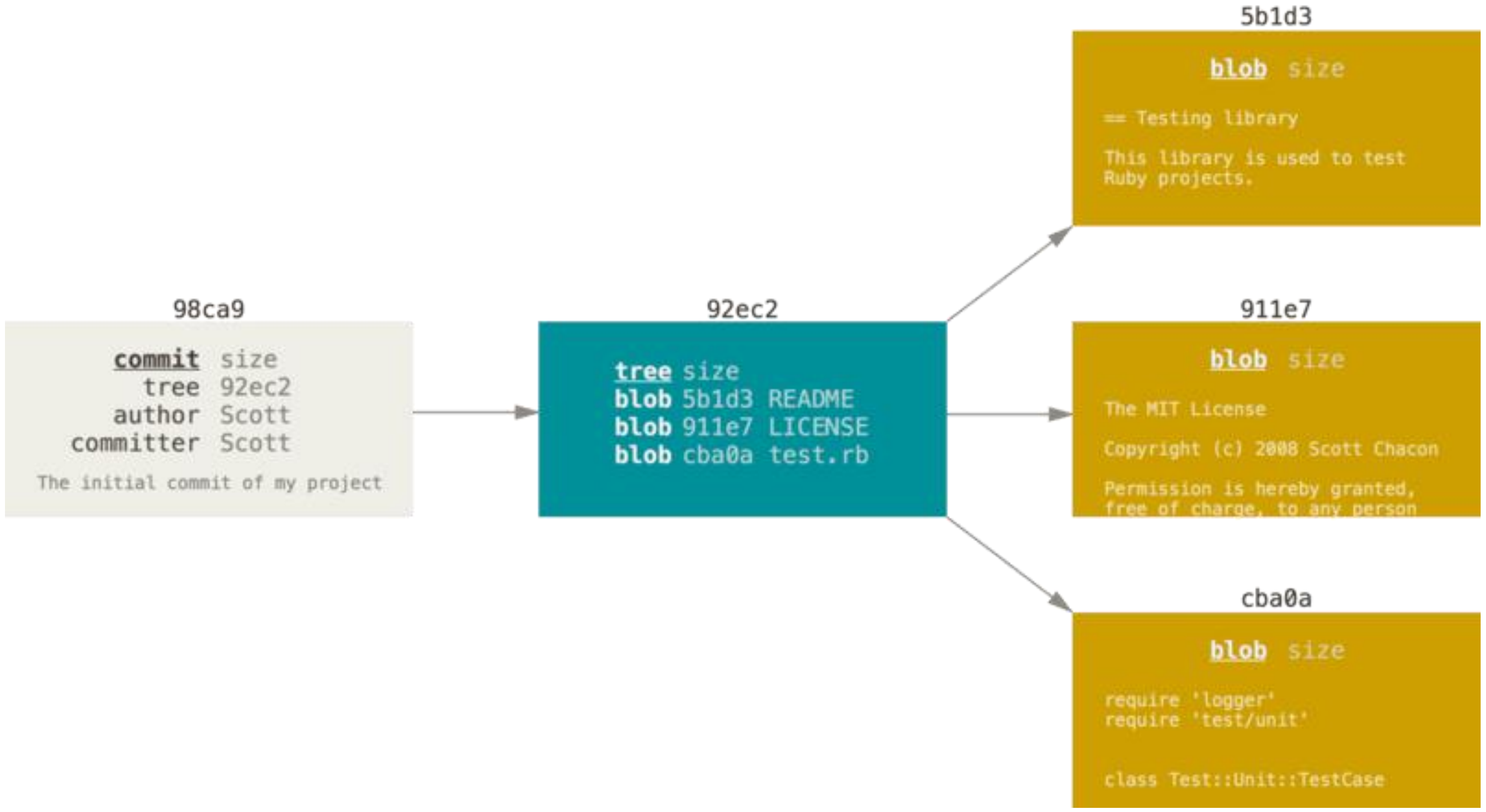
Example: The seven rules of a great Git commit message

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

Looking at history

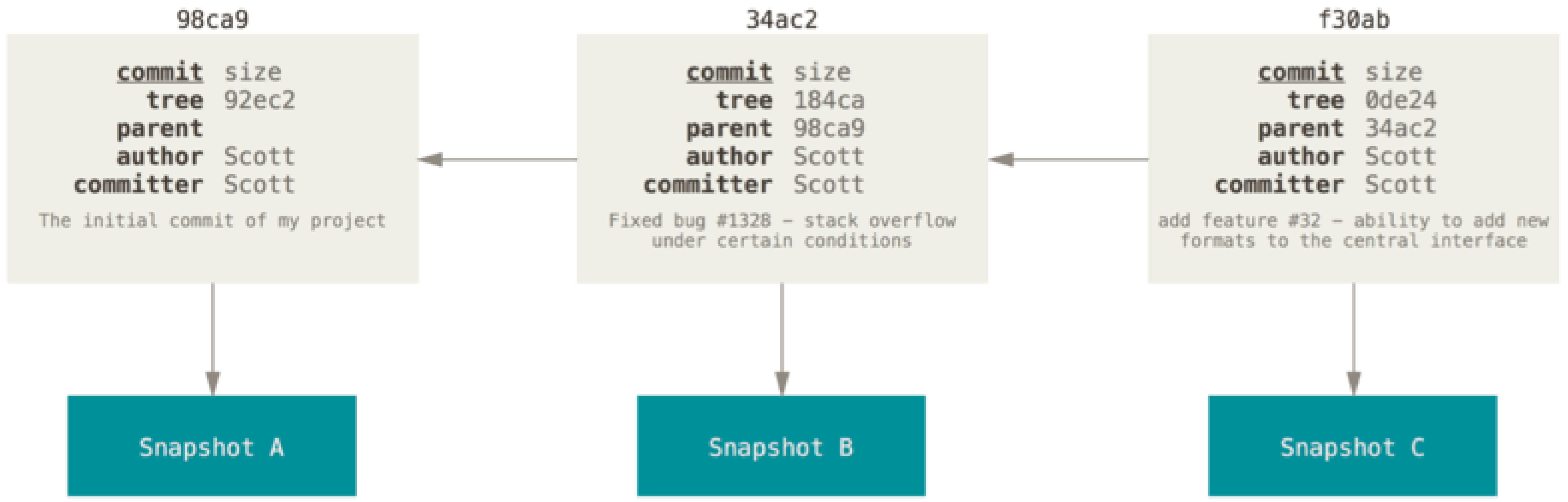
```
$ git log -[N]
$ git log --oneline
$ git log --oneline --graph
```

Files and the commit history



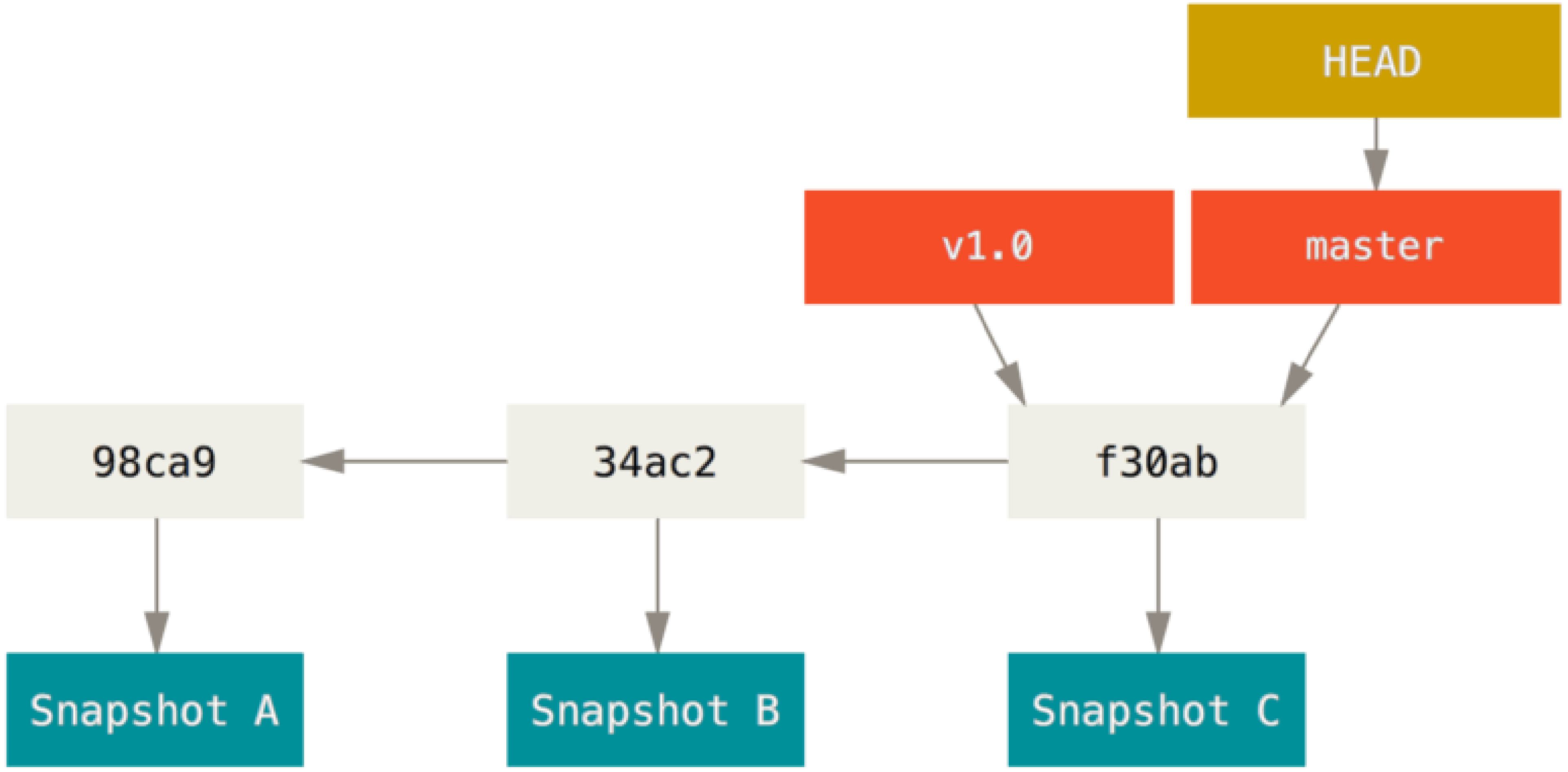
Images taken from the Pro Git book -- freely available online and recommended for further reading
https://git-scm.com/book/en/v2

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

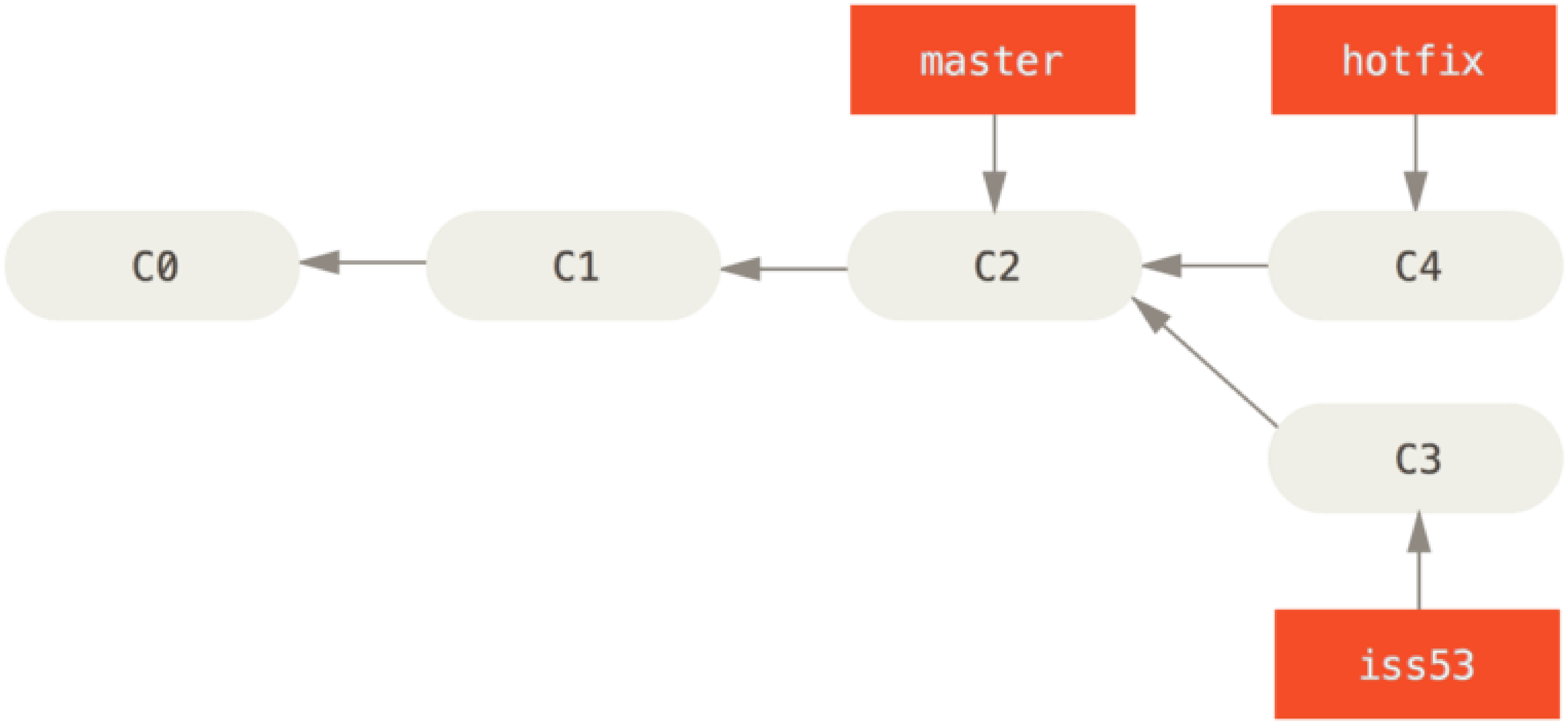


Images taken from the Pro Git book -- freely available online and recommended for further reading
https://git-scm.com/book/en/v2

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!

Say that a user has made errors when editing their current version of analysis.py and wants to recover the last committed version of the file. Which command should she use?

1. $ git checkout HEAD
2. $ git checkout HEAD analysis.py
3. $ git checkout HEAD~1 analysis.py
4. $ git checkout <unique ID of last commit> analysis.py
5. Both 2 and 4

Pulling some ideas together – what gets printed to the screen?

```
$ echo "Venus is beautiful" > venus.txt

$ git add venus.txt

$ echo "Venus is too hot to be suitable as a planetary base" >> venus.txt

$ git commit -m "Comment on Venus as an unsuitable base"

$ git checkout HEAD venus.txt

$ cat venus.txt #this will print the contents of venus.txt to the screen
```

Next week in Part 2:
Looking at the feature-rich capabilities of GitHub!!

Any Future Questions:
Email me at bwinjum@oarc.ucla.edu