Software Engineering Tools

JD Kilgallin

CPSC:480

09/07/22

GitHub docs

Finding the right tool is often half the battle –Andy Rooney

Notes

- Missing a quiz due to communicable illness symptoms; updated policy
  - If you're not able to take the quiz that day at all, I need a doctor's note.
  - If you just don't want to get others sick, email me at least one hour before and plan to take the quiz at home, on-camera, via Microsoft Teams (or Panopto, tbd) at exactly 5:15.
- BrightSpace submissions are open for project 1, GitHub username, and project 2 (first group project) teams
- Project 1 – What can go in a portfolio?
  - No standard projects like CS II. If you're not sure about a project, **email me.**
  - Nothing obscene, discriminatory, or defamatory (read GitHub ToS).
  - Nothing that violates a patent or trademark.
  - If you have a robust portfolio, you can consider omitting your weakest work.

What if I don't have anything for a portfolio?

- Option A – Make something! Practice a new language or framework and solve a small problem:
  - Something from LeetCode or other interview question database
  - Small GUI or CLI game or utility (e.g. tic tac toe, wordle, calculator, dice roller)
  - Intermediate math problem (combinatorics, RSA key generator, calculus)
- Option B – Find an open source project with an open issue and contribute a fix or enhancement. Fork the project and send a PR.
- Option C – Identify three projects that you would *like* to have in your portfolio by the end of the year. Submit a **detailed** report on each stating exactly what the project would be, how you would build it, & what skills you need to gain/obstacles you need to overcome. *A or B should be easier*.

Learning Objectives

- Types of tools involved in software engineering
- Features of project management tools
- Features of development tools
- Features of testing tools
- Features of tools for builds, releases, and version control
- Using Git
- Using GitHub
- Using Markdown

Software engineering tools

- Almost entirely consists of *other* software, including:
  - General-purpose software (e.g. web browser, Word, Adobe Acrobat)
  - Business software (e.g. Excel, SalesForce, learning management system)
  - Communication software (e.g. Outlook, Slack, Zoom)
  - Engineering/project software (e.g. Trello, Microsoft Visio)
  - Systems administration/operation software (e.g. VMWare, PuTTY, kubectl)
  - Developer software (e.g. Visual Studio, notepad++, gcc, git)
  - Domain-specific software (e.g. Postman/Fiddler for Internet systems)
  - Company software (most companies "dogfood" whatever they sell)
- Hardware – desktop, laptop, multiple monitors, high-end keyboard
- Whiteboard, smartphone, books, chair/desk, coffee/snacks

Project management tools

- Track the work curently in progress, completed, and still to be started.
- Hierarchical structure of projects, features, items, and tasks.
- Title, description, priorities, history, product/area, target product versions/dates, time estimate, supporting files, comments.
- Current status, personnel assignment, and *definition of done.*
- Bug reports, root cause analysis, severity and impact notes, criteria for fix.
- Links to parent/child items, progress-blocked-by, duplicates, related items.
- May incorporate source code, code reviews, test cases, and relationship between code checkins/authors, tests/signoff, bugs, original work items.
- Common tools include Trello, Azure DevOps, JIRA, Bugzilla.

Integrated Development Environment (IDE)

- Editor – syntax highlighting, error detection, auto-complete, navigation.
- Solution Explorer – View and manage project components & relationships.
- Package manager – Find/add/update needed libraries, other dependencies.
- Compiler – Build the project locally for target architecture & framework.
- Debugger – Execute the built project with ability to pause, examine state.
- Source control – Pull and push changes from central location/other devs.
- Plugins, automation, GUI designers, endless bells and whistles.
- Examples: Android Studio, Atom, BlueJ, Eclipse, Idle, IntelliJ, Light Table, NetBeans, Visual Studio, Visual Studio Code, Xamarin, XCode.

Command-line development

- A *shell* is an interface to a system. As with other software, can either use a *Command-Line Interface (CLI)* or *Graphical User Interface (GUI).* Major CLI shells are PowerShell on Windows and GNU Bash cross-platform.
- Command-line systems have a higher learning curve, but are very expressive, easy to use remotely (even over slow connections), easy to compose and integrate, and amenable to scripting automated processes.
- Lack of mouseclicks/moving hands can greatly speed activity, especially in combination with expressive tools and automated scripts.
- Free and Open-Source Software toolchains frequently emphasize CLI use.
- Popular command-line code editors include vim, emacs, and "alternatives".

Virtualization

- A system optimized for software development is different than a system optimized for general business use.
  - May require different operating system or other system software.
  - Developer needs administrator privileges, maybe even domain admin.
  - May need to perform unsafe operations that could risk equipment and data.
  - May need network communications that corporate IT would typically restrict.
  - May need to restart frequently or revert to older system snapshots.
  - Hard use may lead to system instability and an eventual need to start over.
  - May need multiple systems for different components or test scenarios.
  - May need to preserve environment across multiple devices.
- Solution is to run a computer within a computer – a *virtual machine*
- Host system runs business applications, plus e.g. VMWare or Hyper-V
- VM can be run in a sandbox and moved to a different host system.

Accessing foreign systems

- Dev tools and software being developed can't always be run locally.
- *Secure Shell Protocol (SSH)* allows access to CLI shell on remote system
- *Remote Desktop Protocol (RDP)* allows GUI shell on remote system.
- *File Transfer Protocol (FTP)* allows up/download on arbitrary systems
- May even want an entire local copy of a remote machine, but VM files are large and potentially difficult to deploy/configure, and may violate software licenses.
- *Containerized* applications and infrastructure-as-code allow this – container uses host OS kernel & includes minimal necessary software

Information tools for development

- *Don't reinvent the wheel*
  - Every major language has an extensive set of libraries to accomplish any kind of common task.
  - Package managers make installation easy, e.g. npm, pip, NuGet, Maven, Go.
  - Good libraries will have documentation and usage samples.
  - Stackoverflow has solution patterns for most common problems, including usage of library code.
  - Build a suite of utilities you find yourself reusing (e.g. wrapping web calls).
- Understand your application
  - Get familiar with debugging all parts of your software and how to examine program state and control flow.
  - Logging is an art, and you are the primary patron.
  - A *profiler* is like a debugger, but focused on analyzing where the program spends most of its CPU/memory/disk IO/network resources.

Testing tools

- *Unit tests* (should) cover individual code paths of a single method.
- *Integration tests* cover interactions between methods/components.
- *End-to-end tests* cover all steps of realistic user activities.
- *Mock* components are used to simulate functionality of components not under test (e.g. returning data from a database or web service).
- Developers usually write unit tests with code, QA Engineers do the rest.
- Test software may be part of IDE, version control, standalone (e.g. Selenium), developed in-house for specific needs, or a combination.
- Selected tests may run automatically each checkin, full set daily w/report.
- Some tools test GUI components by emulating mouse motion/clicks.
- Some tools allow *static analysis* (search for common bugs or vulnerabilities like SQL injection). May use external assessment firm specializing in this.

Software Configuration Management (SCM)

- Actually pre-dates software! Programs using wired plugboards (e.g. ENIAC) need record of cable configuration and a log of changes to it.
- SCM systems primarily track revisions to source code. "Version control", "source control", and "repository" *may* be interchanged, but technically those only refer to one component of SCM.
- SCM is the primary means of collaboration between coders.
- Modern SCM incorporates code reviews, automated builds, test execution, checkin signoff procedures, defect tracking, etc.
- SCM systems track released versions, so that prior releases are accessible, and critical patches can be applied to very old versions.

Version Control/SCM

- First tool *Source Code Control System (SCCS)* 1972 for UNIX.
- *Revision Control System (RCS)* 1982 from Purdue, now part of GNU.
- *Concurrent Versions System (CVS)* 1986 on top of RCS. Still in use.
- *Subversion (SVN)* developed to address unfixable issues with CVS.
- *Perforce* used in large-scale projects, incl Microsoft, Google, SAP.
- *Git* developed for Linux; distributed, scalable, non-linear development
- *GitHub* is a popular online host for Git repositories & related content.
- *DockerHub* is a registry of versioned containers for DevOps and Go.
- Some tools like Azure DevOps offer both SCM & project management.

Git – repositories

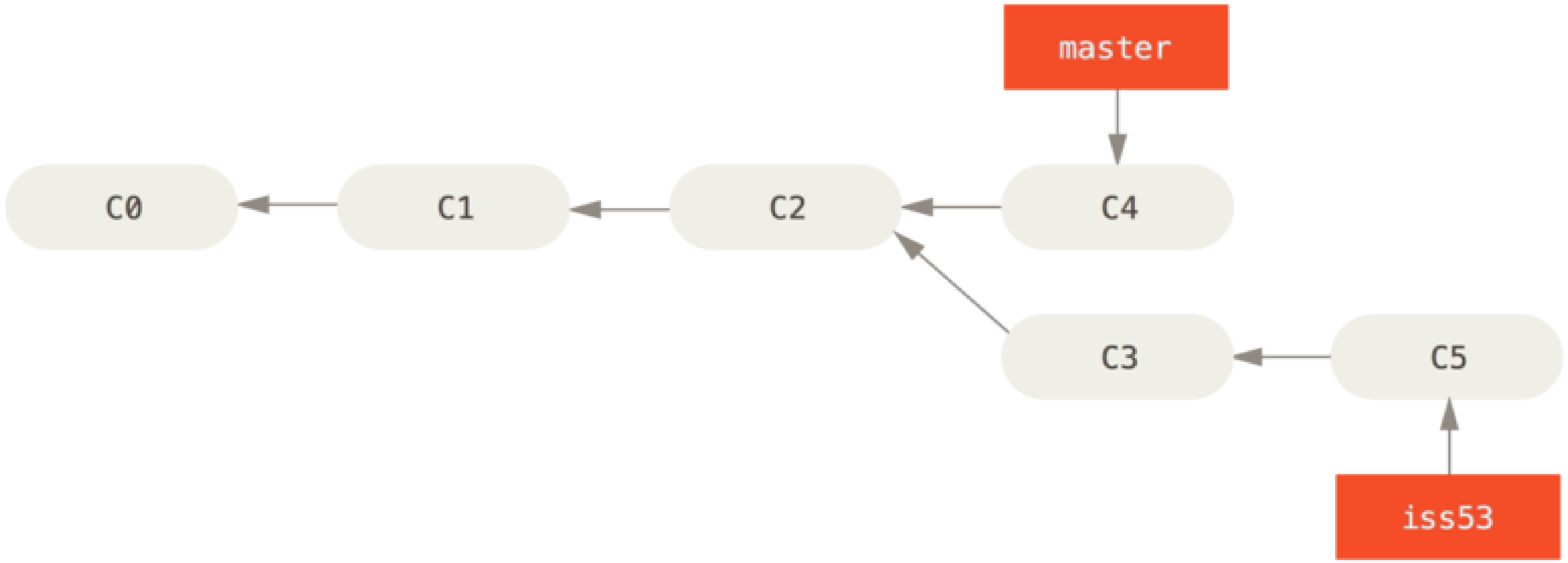
- SCM system built to record changes to files. Allows users to merge simultaneous changes, revert changes, recover old versions, see who changed something and when, compare files at different times, etc.
- A *repository* is a directory/folder representing a project and tracks changes to files and subdirectories. Always has a ".git" directory.
- A repository can exist solely on one machine, or it can be shared, in which case multiple copies can be *cloned* from the *origin* copy.
- While git can be used without a central repository, typically each developer working on a single project will clone a copy from a central source hosted and managed by the company or product owner.
- Git is very complex and we will cover only a subset of features, commands, and options needed to support core use cases.

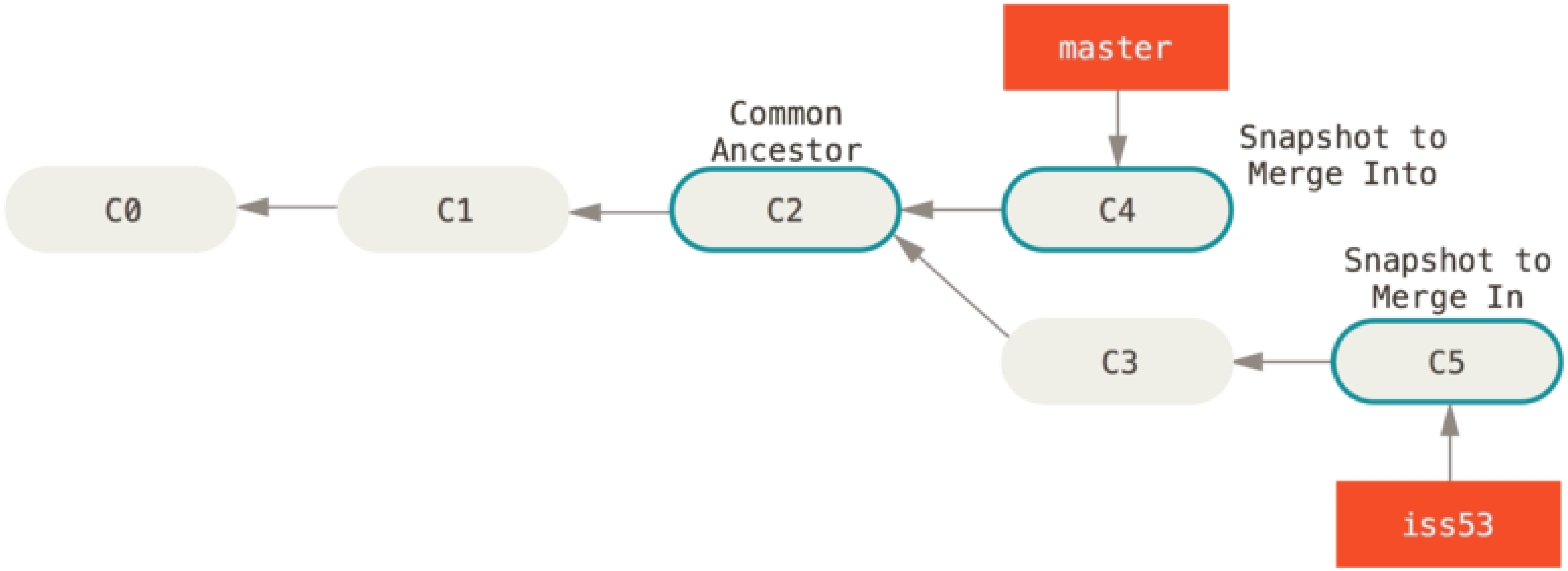
Git – commits

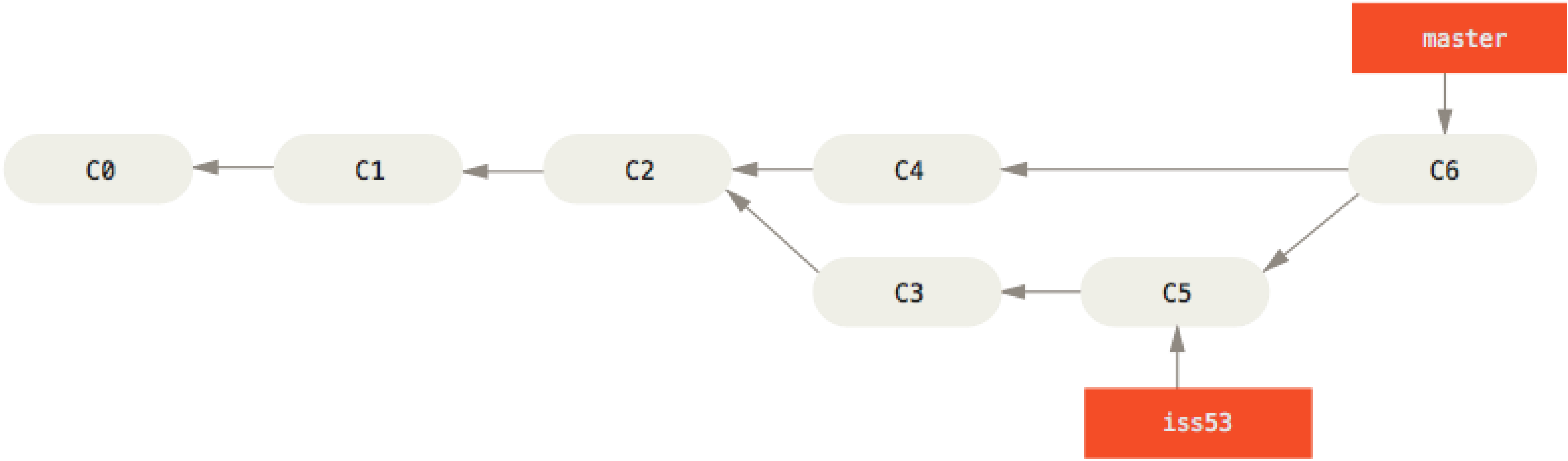
- A *working tree* is a local copy of a git repository where changes are made. Changed files are *staged* for a tracked update, and staged files are *committed*, creating a new local snapshot of the repository.
- New snapshots are *pushed* to origin, and others' changes are *pulled*. Any remote changes must be pulled/merged before yours can be pushed.
- Git stores each commit with a snapshot of changed files, metadata like author and message/title, and pointer to parent commit(s).
- A commit is identified by a hash of the snapshot. You can always checkout a commit to restore the local copy to a previous state.
- A *tag* can be created that serves as a **static** pointer to a specific commit. The main use is to tag a release with the version number.

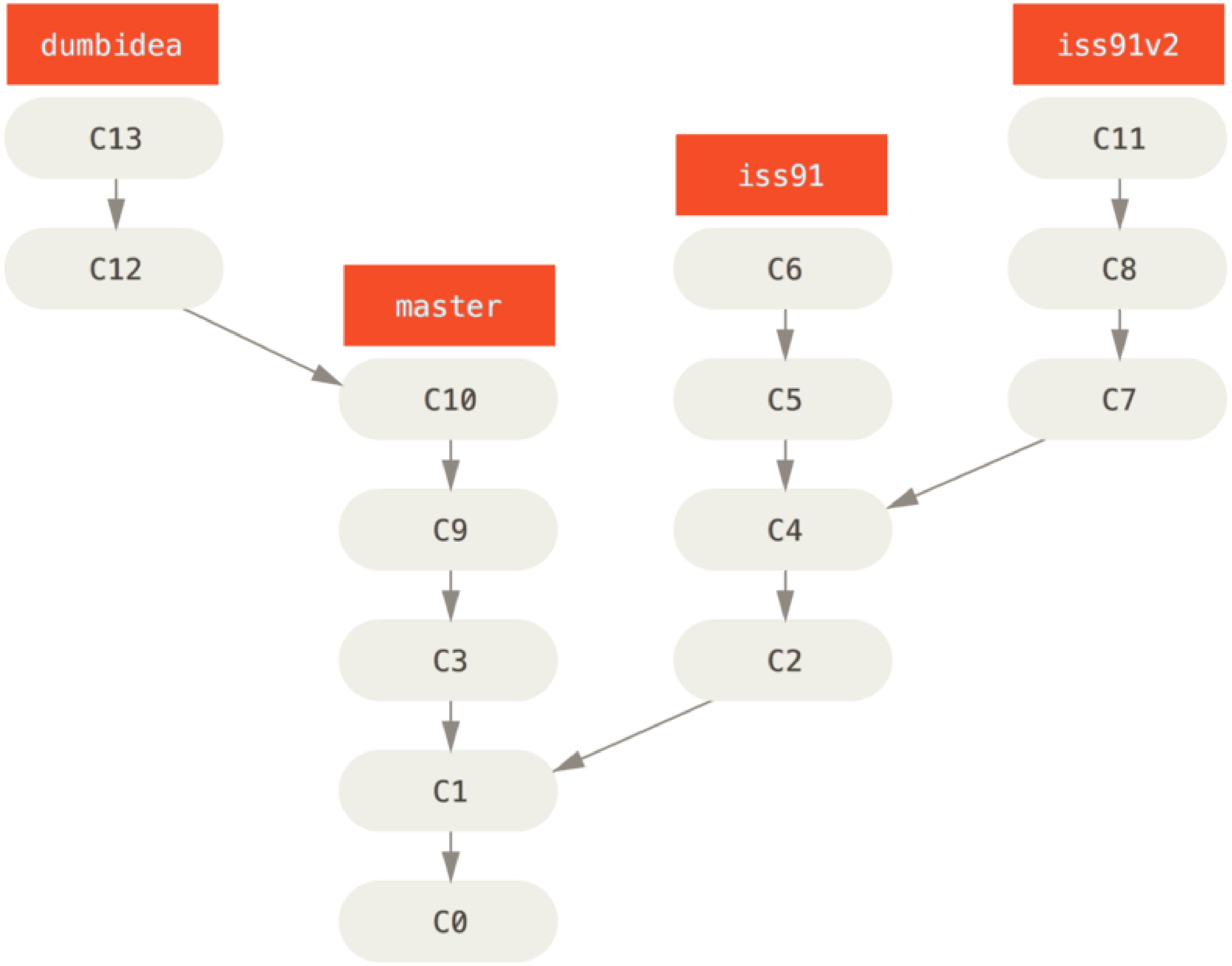
Git – branches

- A *branch* is a pointer to a commit **that moves** with new commits. "master" or "main" is the default branch.
- HEAD is a local pointer to the commit you're currently working from and moves when you switch branches.
- Branches, like repositories, can be local-only or shared.
- Branches are created to develop specific features, releases, bug fixes, experiments, personal notes, or other reasons.
- Branches or clones can be *merged*, creating a new commit. Conflicting changes must be resolved. Many tools can (partially) automate this.


master
C0
C1
C2
C4
C3
C5
iss53


master
Common Ancestor
Snapshot to Merge Into
C0
C1
C2
C4
Snapshot to Merge In
C3
C5
iss53


master
C0
C1
C2
C4
C6
C3
C5
iss53


dumbidea
C13
C12
master
C10
C9
C3
C1
C0
iss91
C6
C5
C4
C2
iss91v2
C11
C8
C7

Git – forks and pull requests

- A *fork* is a new repository with a reference to the original repository. This is different from a branch in that ownership and permissions are managed by the person forking, and not the original repo owner.
- A *pull request* is a request to merge changes from a branch **or** a fork into another, usually the original main branch. These must be approved by someone authorized to modify the destination repo.
- Most OSS limits who can push, or approve pull requests, so changes are made by forking, then issuing a pull request with desired changes.
- Even within a single repo, common practice is for development to happen in a separate branch, and code reviews are done via pull request to main.

Git – basic usage

- Git is usually used from the command line (mainly Linux/bash), but GUIs exist. Commands must be run from within a repository directory.
- git init – create a new local repo from current directory
- git clone <URL> – create local copy of repository accessible at URL.
- git status – Show tracked-vs-untracked & modified-vs-unmodified files
- git add * – Stage all untracked files for commit
- git commit -a -m "Commit message here" – Create local snapshot with newly-tracked files & modified/deleted files previously tracked.
- git pull – Download new commits that were pushed to central host.
- git push – Upload local commit(s) to central repository host
- git log – See commit history

Git – more usage

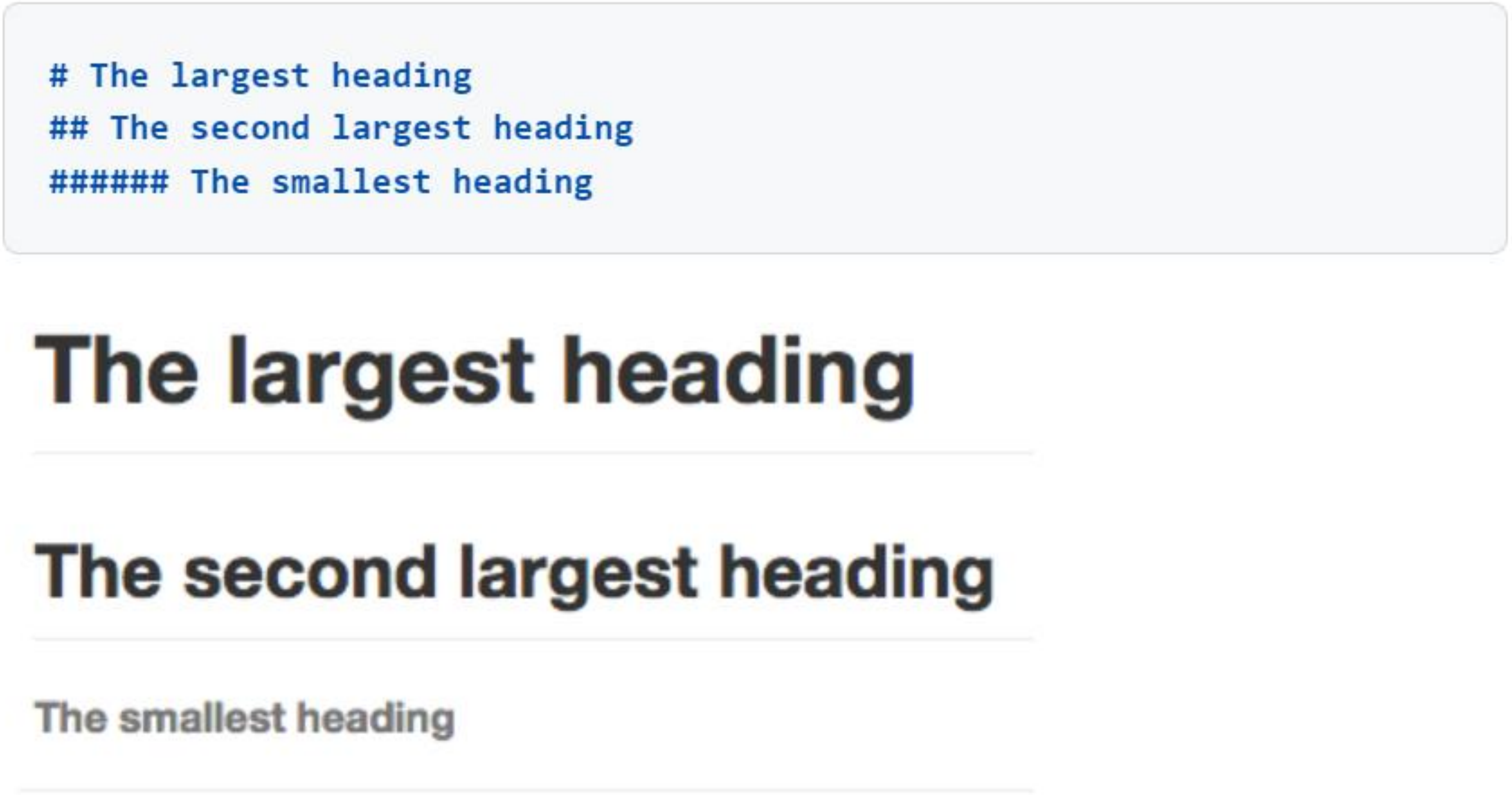
- git branch foo – Create new branch named foo.
- git branch – Show existing branches and the active branch.
- git checkout foo – Switch HEAD and local directory contents to last commit in foo branch. git checkout main to switch to main branch.
- git checkout – Revert local repo to last commit in current branch.
- git diff – Compare files between two commits, tags, branches, or current state. *Many* different options.
- git merge foo – Merge commits in foo branch into active branch.
- git mv oldFilename newFilename – Rename file and preserve tracking.
- git <command> --help – Show usage of command. See References slide for more guides from GitHub, Atlassian, & official *Pro Git* book.

GitHub

- Model is one account per user, multiple repositories per account.
- Free *public* and *private* repos. Anyone can clone or fork public repos, but owner limits who can push changes, issue pull requests, do other tasks.
- Repos (and accounts) can also be associated with an organization. Organizations can centrally manage accounts, groups, permissions, etc
- GitHub allows for online documentation in *Markdown* & even online edits.
- A *release* can be issued from a particular commit in a repository to make a versioned binary package.
- *GitHub Actions* allow automation to occur for checkin, release, etc.
- *GitHub Issues* allows for some project management and defect tracking.

Markdown

- Lightweight, easy-to-use markup language for text formatting.
- More concise and easy to write than alternatives like HTML
- Variants supported in Reddit, Teams, Discord, wikis, Stackoverflow
- Not well standardized, but basic features common in most places.
- May hijack attempts to write plain text (e.g. underscores, line breaks)
- Many tools let you see a preview or auto-replace with formatted text.
- Supports font and paragraph formatting; structures like lists, bullets, and tables; graphical content like diagrams, images, equations, and emoji; links and references; etc.



```
# The largest heading
## The second largest heading
###### The smallest heading
```

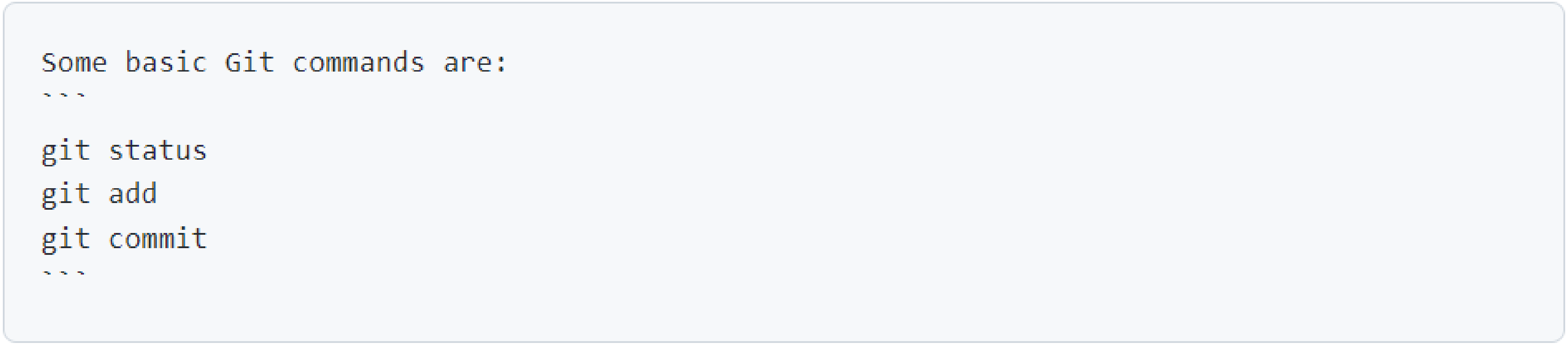
The largest heading

The second largest heading

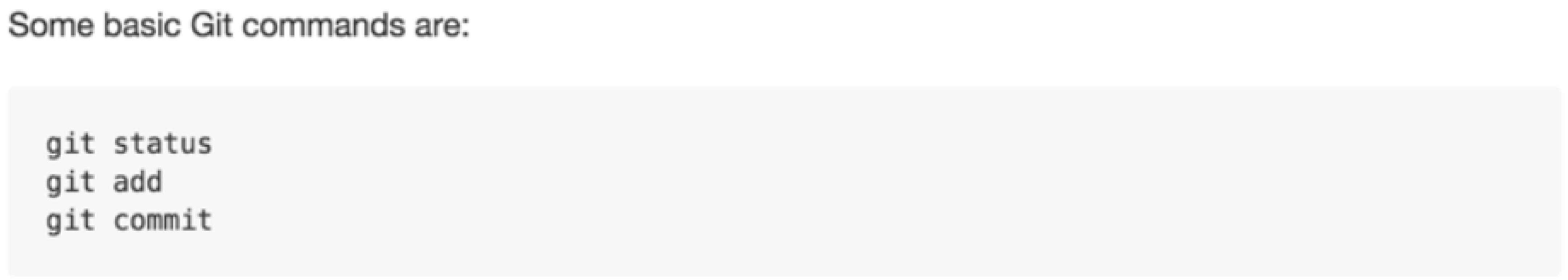
The smallest heading

Bold	`** **` or `__ __`	`**This is bold text**`	**This is bold text**
Italic	`* *` or `_ _`	`*This text is italicized*`	*This text is italicized*
Strikethrough	`~~ ~~`	`~~This was mistaken text~~`	~~This was mistaken text~~
Subscript	`<sub> </sub>`	`<sub>This is a subscript text</sub>`	This is a subscript text
Superscript	`<sup> </sup>`	`<sup>This is a superscript text</sup>`	This is a superscript text

To format code or text into its own distinct block, use triple backticks.

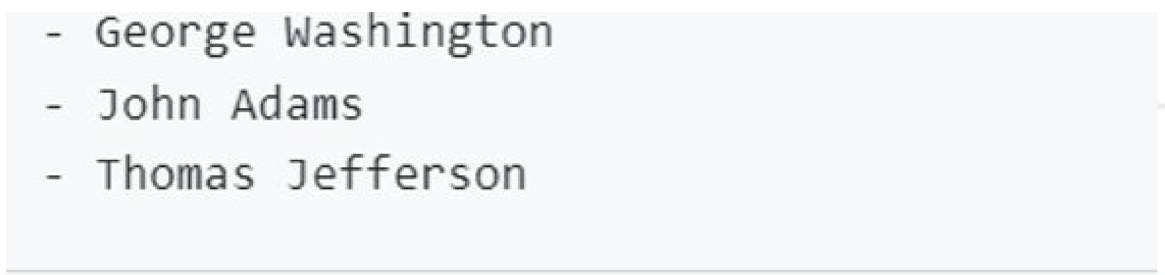


~~~
Some basic Git commands are:
```
git status
git add
git commit
```
~~~

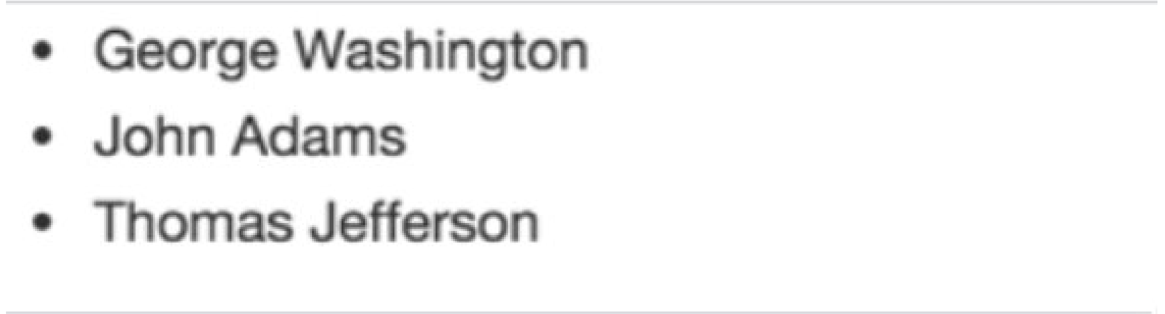


Some basic Git commands are:

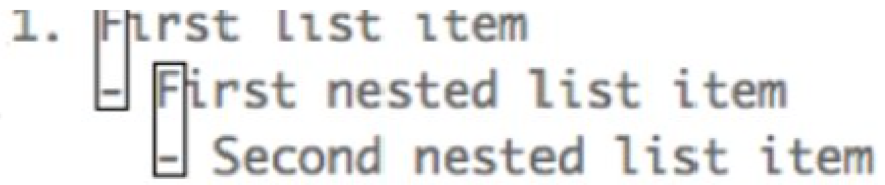
```
git status
git add
git commit
```



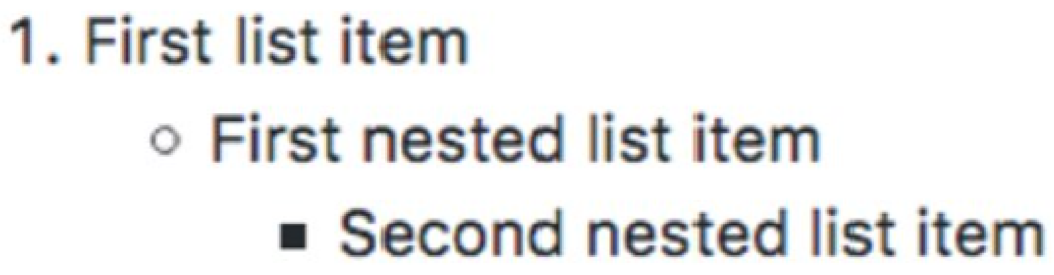
```
- George Washington
- John Adams
- Thomas Jefferson
```



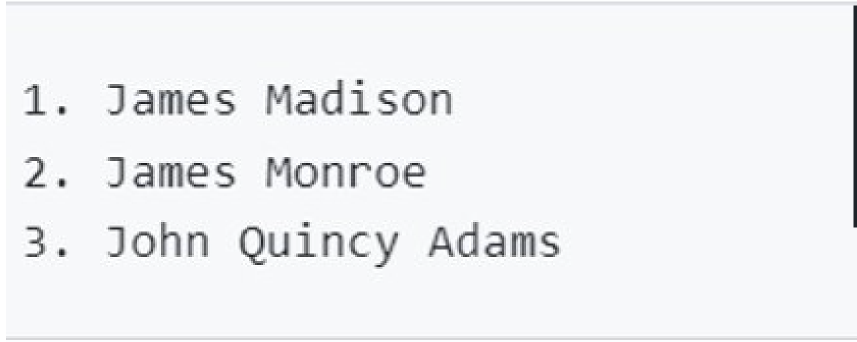
- George Washington
- John Adams
- Thomas Jefferson



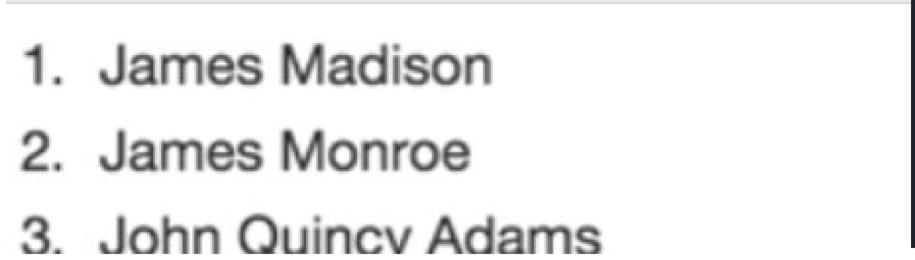
```
1. First list item
   - First nested list item
     - Second nested list item
```



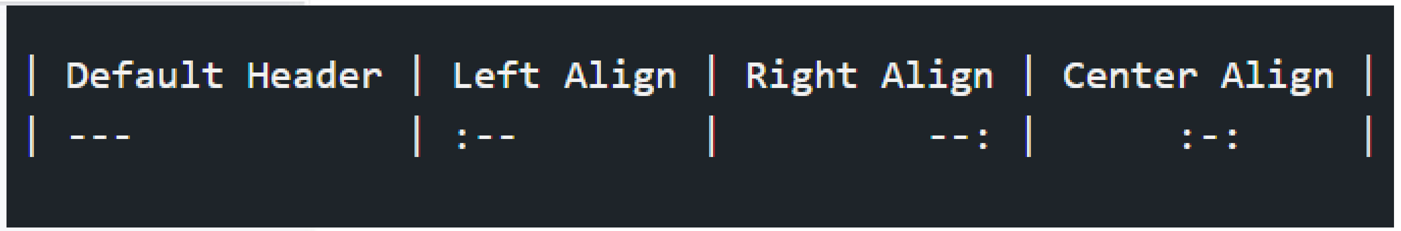
1. First list item
   - First nested list item
     - Second nested list item



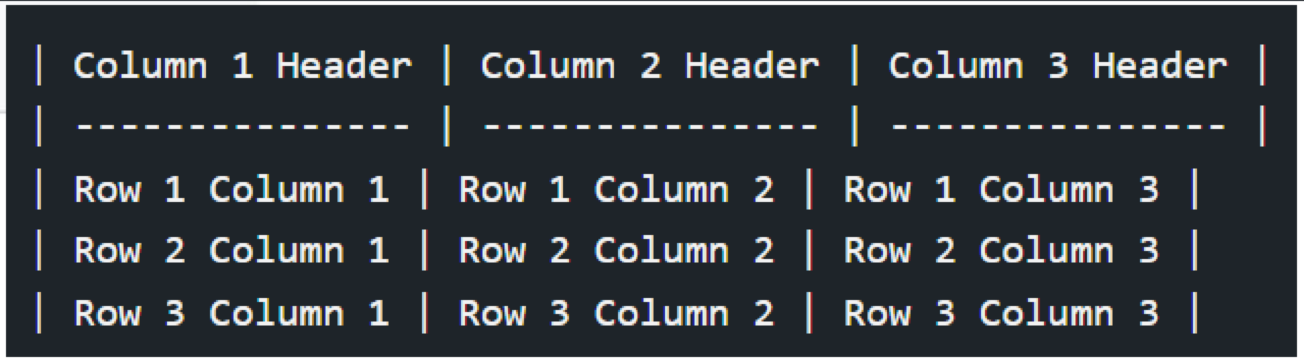
```
1. James Madison
2. James Monroe
3. John Quincy Adams
```



1. James Madison
2. James Monroe
3. John Quincy Adams



```
| Default Header | Left Align | Right Align | Center Align |
| ---            | :--        |         --: |     :-:      |
```



```
| Column 1 Header | Column 2 Header | Column 3 Header |
| --------------- | --------------- | --------------- |
| Row 1 Column 1 | Row 1 Column 2 | Row 1 Column 3 |
| Row 2 Column 1 | Row 2 Column 2 | Row 2 Column 3 |
| Row 3 Column 1 | Row 3 Column 2 | Row 3 Column 3 |
This site was built using [GitHub Pages](https://pages.github.com/).
```

This site was built using GitHub Pages.

A relative link is a link that is relative to the current file. For example, if you have a README file in root of your repository, and you have another file in *docs/CONTRIBUTING.md*, the relative link to *CONTRIBUTING.md* in your README might look like this:

```
[Contribution guidelines for this project](docs/CONTRIBUTING.md)
```

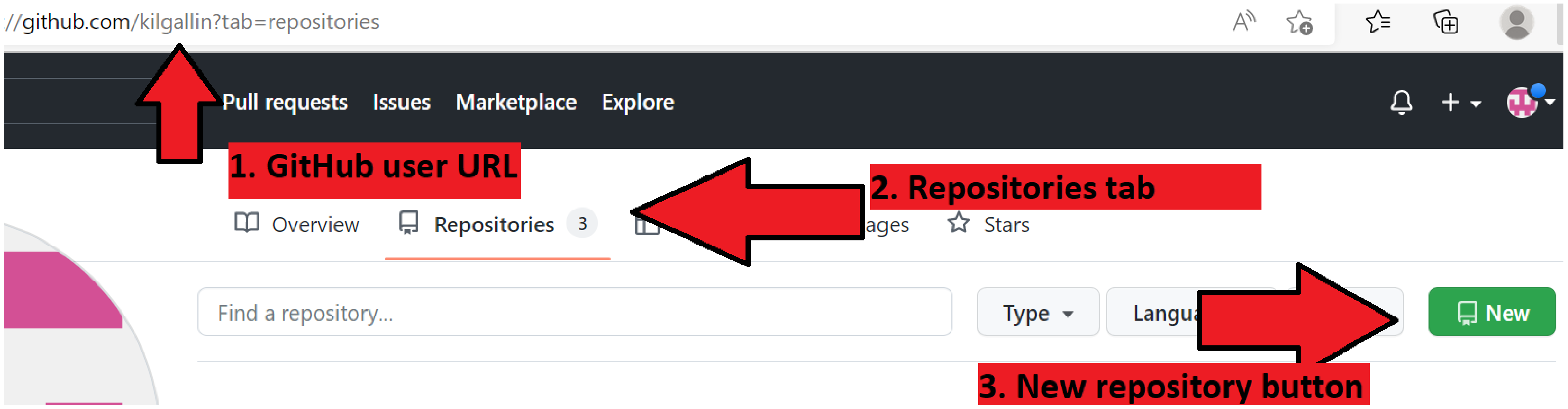
```
![This is an image](https://myoctocat.com/assets/images/base-octocat.svg)
```

Markdown – not shown

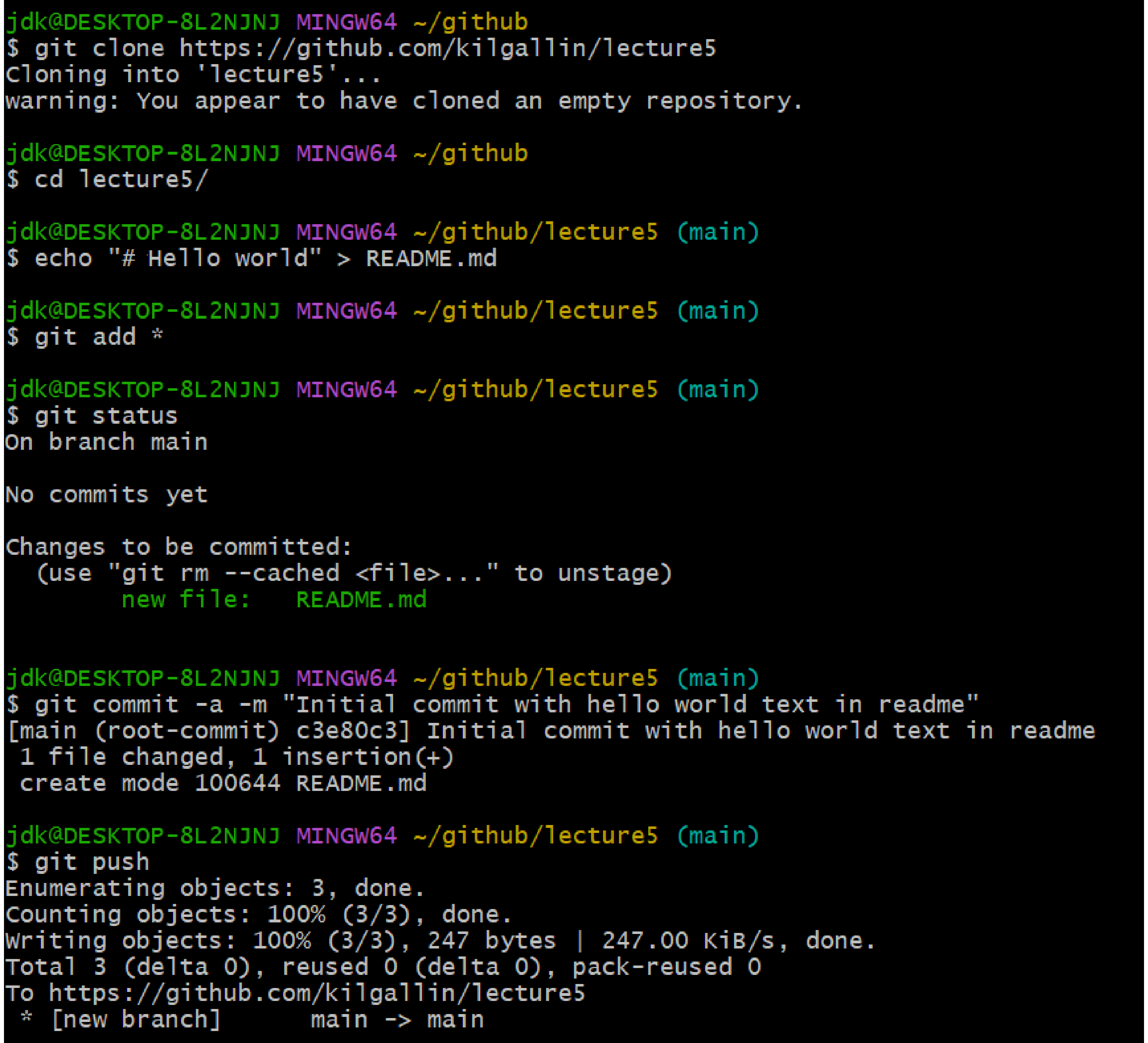
- Paragraph and line break formatting
- Automated table of contents
- Quotation blocks
- Relative links
- Color
- @Mentions
- Emoji
- Escape sequences
- Diagrams
- LaTeX (typesetting equations)
- Footnotes
- More!

Git/GitHub "Hello World"

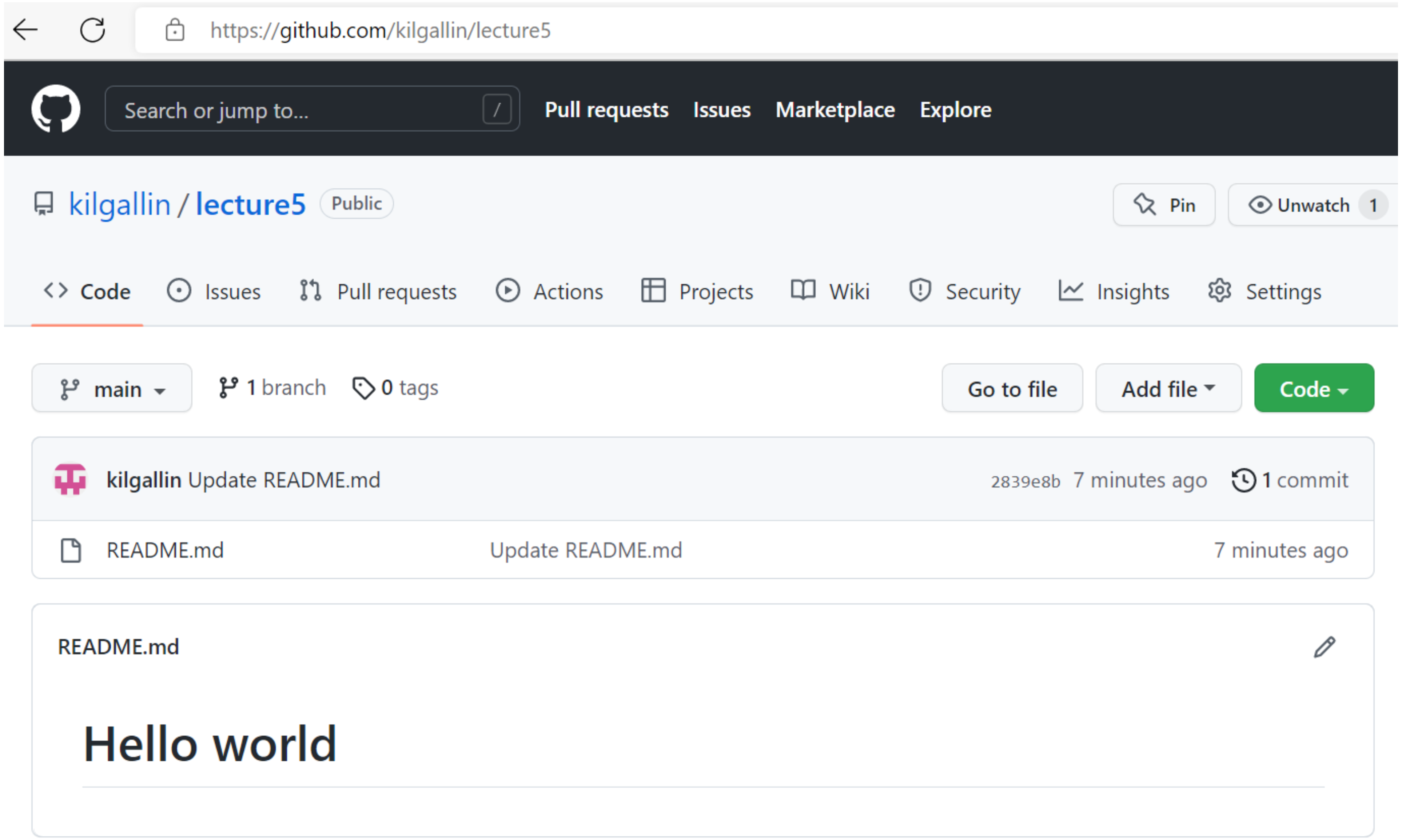
- Create a repo in GitHub.
- Clone a local copy
- Make a change (Add README in markdown)
- Commit the change
- Push the change to GitHub
- View the change in GitHub



- Access GitHub.com in a browser (There will be a "New" shortcut here)
- Click user icon and go to "Your profile". You can also use direct URL (1)
- Go to "Repositories" (2) & click "New" (3), or use homepage shortcut.
- Enter a repository name and description and click "Create repository"
- Copy URL, paste into "git clone" command, & change to new directory
- Run git commands to commit and push markdown in README.md.
- Follow directions on "git push" to enter name, email, GitHub creds.
- Navigate to repository in browser or paste URL again.



```
jdk@DESKTOP-8L2NJNJ MINGW64 ~/github
$ git clone https://github.com/kilgallin/lecture5
Cloning into 'lecture5'...
warning: You appear to have cloned an empty repository.

jdk@DESKTOP-8L2NJNJ MINGW64 ~/github
$ cd lecture5/

jdk@DESKTOP-8L2NJNJ MINGW64 ~/github/lecture5 (main)
$ echo "# Hello world" > README.md

jdk@DESKTOP-8L2NJNJ MINGW64 ~/github/lecture5 (main)
$ git add *

jdk@DESKTOP-8L2NJNJ MINGW64 ~/github/lecture5 (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md


jdk@DESKTOP-8L2NJNJ MINGW64 ~/github/lecture5 (main)
$ git commit -a -m "Initial commit with hello world text in readme"
[main (root-commit) c3e80c3] Initial commit with hello world text in readme
 1 file changed, 1 insertion(+)
 create mode 100644 README.md

jdk@DESKTOP-8L2NJNJ MINGW64 ~/github/lecture5 (main)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 247 bytes | 247.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/kilgallin/lecture5
 * [new branch]      main -> main
```



kilgallin Update README.md 2839e8b 7 minutes ago 1 commit

README.md	Update README.md	7 minutes ago

README.md

Hello world

References

- GitHub ToS
- History of software configuration management - Wikipedia
- Download Git
- The Linux command line for beginners
- Setting up a repository | Atlassian Git Tutorial
- Saving changes | Atlassian Git Tutorial
- Set up Git - GitHub Docs
- Hello World - GitHub Docs
- Basic writing and formatting syntax - GitHub Docs
- Official book: Pro Git, 2nd edition. Scott Chacon & Ben Straub. 2014. Apress.

- ***Bring a laptop to next lecture with Git installed, & create GitHub account.***