Introduction to Git and GitHub for Writers

Puget Sound STC Workshop

Peter Gruenbaum

SDK BRIDGE

Introduction

- This Course Covers:
  - What are Web APIs?
  - Markdown
  - JSON
  - What is REST?
  - HTTP Methods
  - Query ParametersTools
  - Next Steps

Schedule

- 8:30 Registration
- 9:00 Introduction
- 9:15 Intro to Git and GitHub
- 10:00 Git add, commit, and push
- 11:45 Break
- 11:00 Git checkout, pull, and merge
- 11:30 Branching

Peter Gruenbaum

- PhD in Applied Physics from Stanford
- Commercial Software Developer
  - Boeing, Microsoft, start-ups
  - C#, C++, Java, JavaScript, Objective-C
- API Writer
  - Brought together writing and technology
  - Since 2003
  - President of SDK Bridge
- Teacher: Programming at middle, high school, and college

SDK BRIDGE

Wi-Fi

- Network: galvanize guest seattle
- Password: none, but requires an email address

- Be sure that you have everything in the Preparation section of the workbook

Download Presentation and Workbook

- Go to http://sdkbridge.com/downloads/
- Enter email to get put on my newsletter list
  - Will notify you of Git/GibHub Udemy course (with coupon!)
  - At most one email every two months
  - Put in a fake email if you don't want me to have it (fake@example.com)

Introduction

Git and GitHub for Writers

What is Git and GitHub?

SDK BRIDGE

Git

- Version control system
- Free
- Open source
- Really fast
- Takes up very little room

GitHub

GitHub

- Commercial platform to host files through Git
- Allows for easy collaboration
- Nice visual user interface
- Free for public files
- Subscription for private files

Why this workshop?

- Writers are being asked to use Git, but…
- Git was designed for developers, not for writers
- Git can be quite complex
- Git is often not intuitive
- Sometimes Git does not have safeguards

- Writers need their own Git and GitHub course!

Notes about this workshop

- We will use the command line rather than a GUI tool
- We will use GitHub, but be aware that other Git platforms exists
- We will write some simple content in Markdown, which is a simple markup language
  - You do not need to know Markdown
- I will show you how to do something, then you will do an exercise where you do it yourself

Docs Like Code

Git and GitHub for Writers

Treating Documentation Like Code

SDK BRIDGE

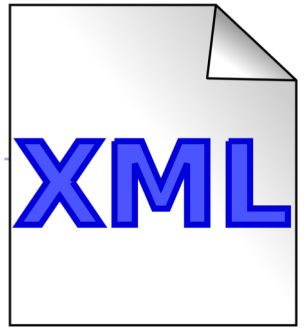
Docs Like Code

- Phrase that means treat writing documents the same way you treat writing code
  - "Docs Like Code" book by writer Anne Gentle
- Documentation in text format, not binary
- Use version control tools (Git, GitHub, etc.)
- Use a review process similar to code reviews

Text vs Binary

- When a document is in a text format, you can open it in a text editor and read it
- Some formats are designed for this:
  - Markdown, reStructuredText
- Some formats are kind of readable
  - XML, HTML
- Binary formats (Microsoft Word files, for example) are not readable at all

XML



- Combination of text and tags
- Used for DITA
- When software is automatically merging two versions of an XML file, it can mess up the structure
- Not ideal for “Docs like code”

Markdown

- Simple markup language
- No tags
  - `**bold**` instead of `<strong>bold</strong>`
- Has many variations (called “flavors”)
- GitHub-flavored Markdown is very popular
- Works well for “Docs like code”

reStructuredText

- Simple markup language
  - A little more complex than Markdown
- Originally came from documenting Python through a tool called Sphynx
- No tags
- Only one variation (no flavors)
- Can extend its functionality
- Works well for “Docs like code”

AsciiDoc

AsciiDoc

- Similar to reStructuredText
- No tags
- Only one variation (no flavors)
- Can extend its functionality
- Works well for “Docs like code”

Git and GitHub for Writers

Version Control

How do you keep track of versions?

SDK BRIDGE

What is version control?

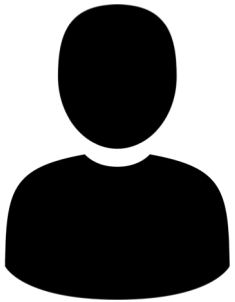
- As you modify a document, you create new versions
- You could give each file a new name:
  - intro-v1.txt
  - intro-v2.txt
  - intro-v3.txt
- Gets confusing when multiple people working on it
- In version control systems, multiple users manage the versions as if they all had the same name
- Also called revision control and source control

Collaboration

- Often you work on a document in a team
- Different people make changes to the document
- Version control systems can help manage this
- Documents are hosted remotely, but worked on locally

Previous version control systems

- Earliest: one person checks out at a time.



Check out

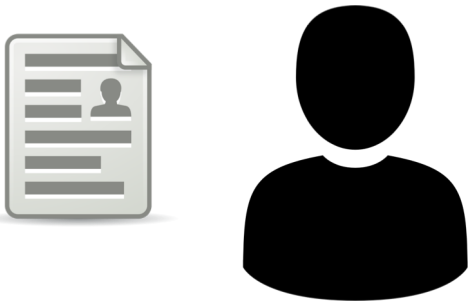
Previous version control systems

- Earliest: one person checks out at a time.

Modify

Previous version control systems

- Earliest: one person checks out at a time.



Check in

Previous version control systems

- Later: multiple people check out simultaneously.

Check out

Previous version control systems

- Later: multiple people check out simultaneously.

Modify

Previous version control systems

- Later: multiple people check out simultaneously.

Check in with Merge

Previous version control systems

- Later: multiple people check out simultaneously.

Check in with Merge

Git

- Created by Linus Torvalds in 2005
- The name doesn't really mean anything
- Rather than storing every version, it stores the differences between versions
  - Much smaller
- Every computer stores all versions
  - Older systems you only stored what you were working on

Other Git innovations

- Git manipulates files using the file system
  - File content changes with git commands
- Separates commit and push
  - Older systems: check in marks and uploads files
  - Git: marking and uploading are two separate things

GitHub and platforms like it

- You can create a git remote repository on any server
- However, commercial platforms make it easier
- GitHub:
  - The most popular

- Others:
  - Provide better security, better integration with other tools, etc.
  - BitBucket, GitLab, Beanstalk, etc.

ATLASSIAN Bitbucket

beanstalk

Git and GitHub for Writers

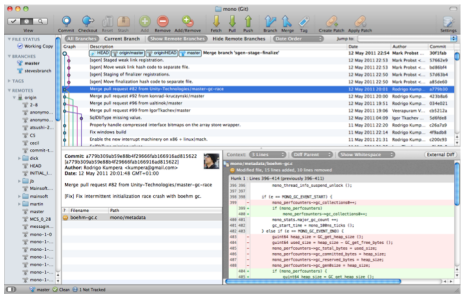
Getting Started with Git

Installing and configuring

SDK BRIDGE

Git Command line vs. GUI

- Command line
  - Type commands in a terminal
  - Gives you full power of git
  - Most people (even non-technical) eventually use command line
- GUI
  - An app with a user interface
  - Not that much less confusing that command line
- This workshop will use the command line only



Mac

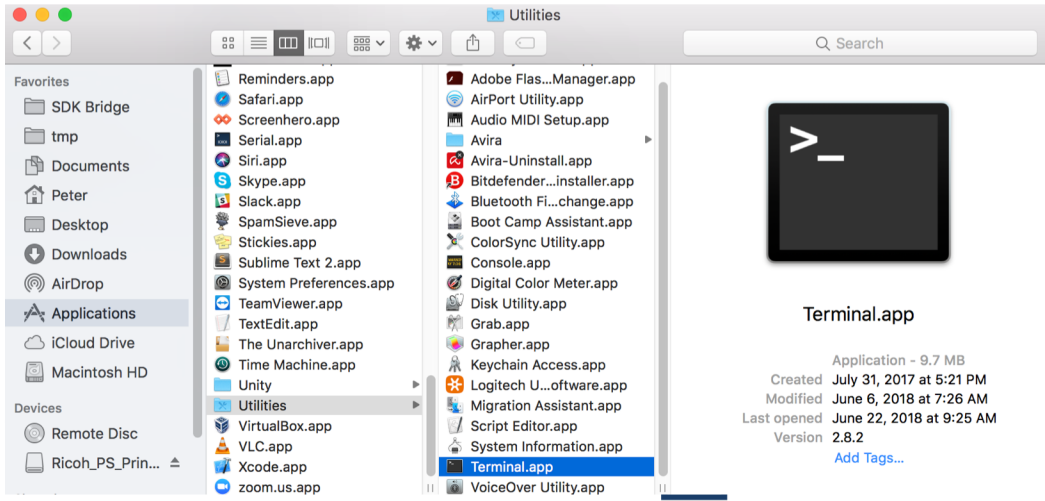
- If you have a Mac, then git is automatically installed already
- All you have to do is use the terminal app
- Can't get easier than that!

Windows

- Will require an installation
- Use special terminal app called “git bash shell”
- Go to https://git-scm.com/downloads
- Click on **Windows**
- Download and install
- Instructions in workbook

Testing to see if git is installed

- The easiest way to test is to have git show you its version
- Mac: Open Terminal app under Utilities
- Windows: Open git bash shell
- Type `git --version`



Git and GitHub for Writers

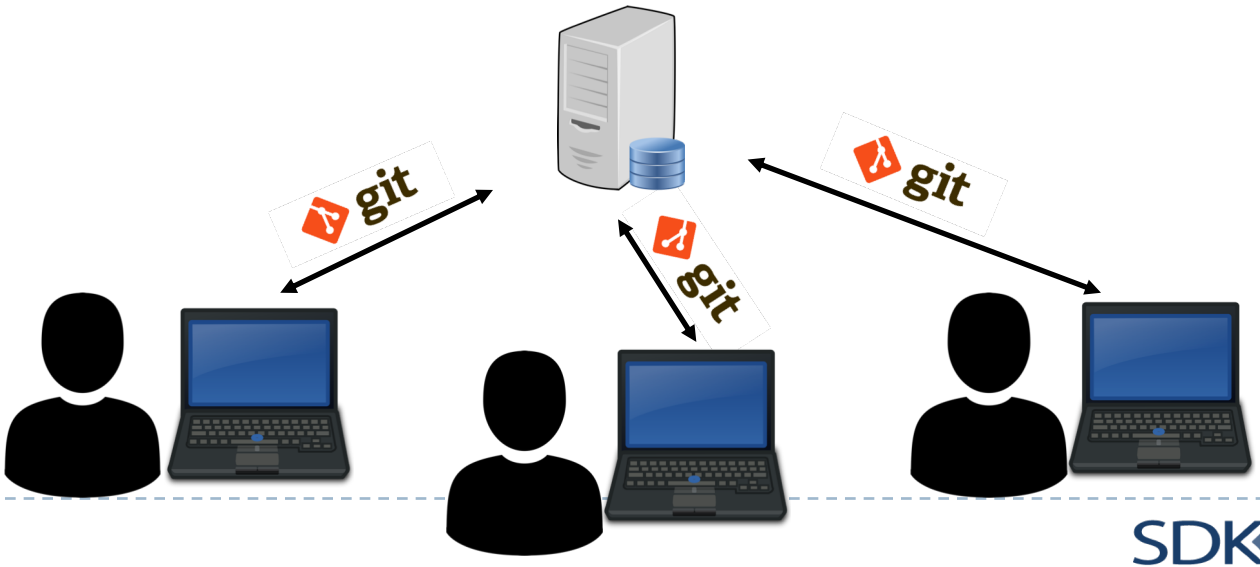
Getting Started with GitHub

Signing up and creating a repository

SDK BRIDGE

What is GitHub?

- A way to host your content remotely
- Multiple users can read and/or write versions
- Everyone uses Git to access it



Creating an account

- Accounts on GitHub are free
- It's free to host files if those files are open to the public to view
- Most companies will pay a subscription for privacy
- Open source projects are often public

Repositories

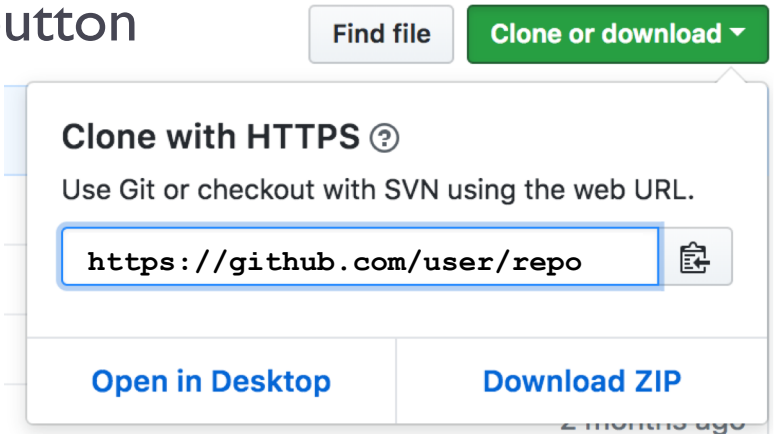
- A place that holds a directory structure (folders) with files
- Contains not just current versions, but all versions and history
- Typically exists both remotely and locally, although not necessarily in sync

Creating a repository

- Can be created locally with Git and then "pushed" to the remote platform (GitHub)
- Easier to create in GitHub and then "clone" it locally

Cloning a repository

- Once repo is created, GitHub shows you its path
  - Click the **Clone or download** button



- Use this in a **git clone** command
  - `git clone https://github.com/user/repo`

Command Line

Git and GitHub for Writers

Typing Git commands

SDK BRIDGE

What is the command line?

- Before 1973, computers did not have GUIs
  - Xerox invented the GUI, and Apple ran with it
- Instead, you interacted with a computer through text
  - Type commands
  - Read responses
- This is called the command line
- You can use git through the Unix operating system command line

Useful Unix commands

- With git, you'll be using Unix commands to move around your directory structure (folders and files)
- **`pwd`** tells you what folder you are in
- **`ls`** tells you what files and folders are in your current folder
- **`cd`** moves you to a new folder
- **`mkdir`** creates a new folder

pwd

- Stands for "**p**rint **w**orking **d**irectory"
  - Folders in Windows and Mac OS are called directories in Unix
- Returns the path to which directory you are working in
- Note that levels are separated with forward slashes, even on Windows

ls

- Short for "**list**"
- Returns a list of all of the files and directories in your current directory

cd

- Short for "**c**hange **d**irectory"
- Changes your current directory to a new directory
- Special directory names:
  - . (one period): your current directory
  - .. (two periods): up one directory
  - ~ (tilde): home directory

mkdir

- Short for "**ma**k**e** **dir**ectory"
- Creates a new directory
- Does not move you into the new directory automatically

Command line shortcuts

- Tab: Fills in the rest of a name
  - Stops if it doesn't have enough information to finish
- Up arrow: Repeats the previous command
  - Multiple up arrows keep going back in history
  - You can then edit the line to make changes

Command structure

- General command structure
  - `command -o --option argument(s)`
- Options modify the command in some way
- One dash for one-letter options
- Two dashes for multiple-letter options
- Options before arguments
- Because elements are separated by spaces, sometimes you need to put arguments in quotes

Example git commands

- Git commands start with git and then a command
  - `git config --list`
  - `git config --global user.name "Peter Gruenbaum"`

- Example one-letter option
  - `git commit -m "Marketing changes"`

Exercise 1: Create a repo and command line

- Follow the instructions in the exercise
- Create a repository in GitHub
- Clone the repository locally

Git Concepts

Git and GitHub for Writers

The four stages of files

SDK BRIDGE

File Stages

- As you work with files in git, they move through four stages

Unstaged Staged Committed Pushed

Stages defined

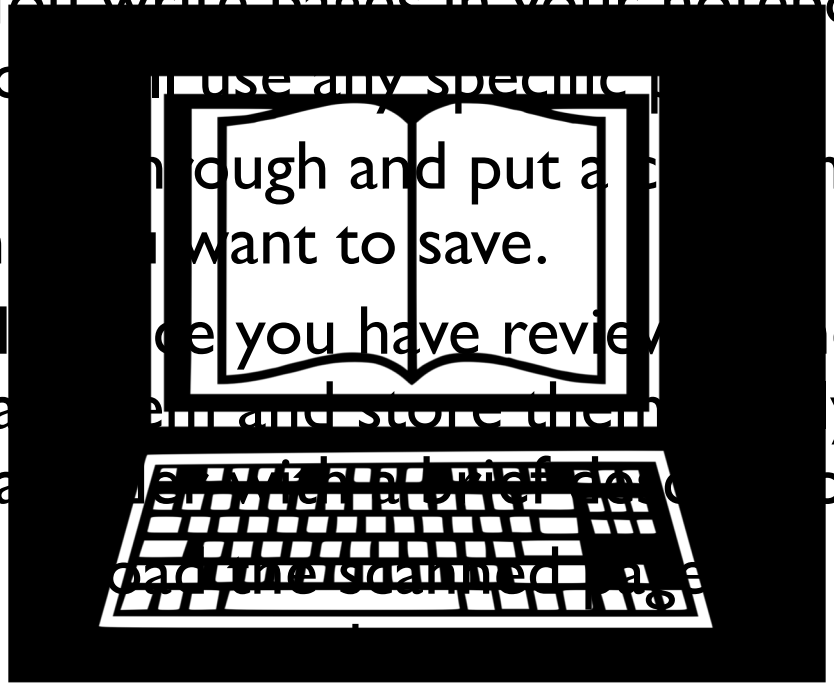
- **Unstaged.** You've made a local change to the file. (Or added or deleted a file.)
- **Staged.** You've marked a local change as something you want to commit to save in git
- **Committed.** You have committed the file, saving this version in git.
  - Each commit has description to help you remember what this version is about
- **Pushed.** You've uploaded the file to the server so that others can access it

When you use stages

- **Unstaged.** You've made additions and changes that you aren't sure you want to keep
- **Staged.** You're pretty sure you want to keep the changes
- **Committed.** You definitely want to keep the changes
- **Pushed.** You want to share the changes with others

Analogy: Writing a book

- **Unstaged.** You write pages in your notebook. You don't know yet if yo[illegible] use any specific [illegible]
- **Staged.** You [illegible]ough and put a [illegible]mark on each page you thin[illegible] want to save.
- **Committed** [illegible]ce you have revie[illegible]e checked pages, you sca[illegible]em and store them [illegible]y on your computer in a [illegible]on.
- **Pushed.** You [illegible]oad the scanned [illegible]a server where other people [illegible]



How to Move Files from One Stage to the Next

- **Unstaged.** Just make changes on your computer
- **Staged.** Use the **`git add`** command
- **Committed.** Use the **`git commit`** command
  - Add a description
  - You don't need to tell it which files, because it commits the staged ones
- **Pushed.** Use the **`git push`** command
  - You don't need to tell it which files because it pushes all the committed files that haven't been pushed already

Text and Binary Files

- Git works on both text and binary files
- Text: documents
- Binary: images, video, audio, etc.
- Both are treated the same
- The only difference: merging different versions
  - Text: software can reliably figure out how to merge
  - Binary: extremely difficult to figure out how to merge

Adding Files to the Repository

Git and GitHub for Writers

How to add a new file

SDK BRIDGE

Text editors

- To do the exercises, you will need a text editor
- Although you can use built-in Notepad (Windows) and TextEdit (Mac), there are better tools available
  - For example, if Git changes the file that's being shown
- Windows: Notepad++
  - Free, powerful
- Mac
  - SublimeText (Free trial)
  - Brackets (Free)

Markdown

- Very simple markup language
- Text-based, very easy to read
- Unlike HTML, no style information
- GitHub can display it nicely
  - GitHub assumes GitHub-flavored Markdown
- We will use it for this course
  - But you really don't need to know it

Useful git commands

- **`git status`** – Provides useful information
- **`git add`** – Moves one or more files from unstaged to staged
- **`git commit`** – Commits any staged files
- **`git push`** – Uploads committed files to GitHub

- Note that to have our file be unstaged, all we have to do is create it!

Placeholder

- Show how to do it

Exercise 2: Create, add, commit, and push

- Follow the instructions in the exercise
- Create a new file
- Stage it
- Commit it
- Push it to GitHub

Changing Files

Git and GitHub for Writers

What happens when you make changes

SDK BRIDGE

What happens when you make a change

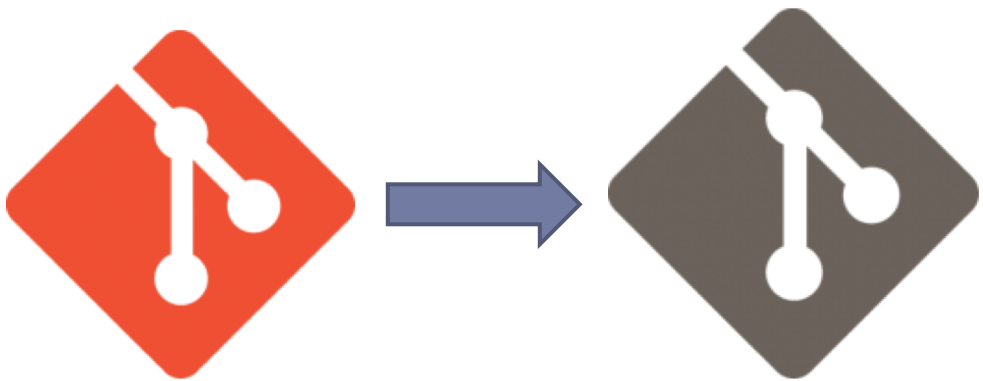
- File is automatically unstaged
- Do a **`git add`** to stage it
- You still have a chance to change it before committing
  - If you do, you will need to do a **`git add`** again to stage it again
- Do a **`git commit`** to commit it
- Do a **`git push`** to upload it to GitHub
- Note that although I will show you how to do this for one file, you can use any of these commands for multiple files

Placeholder

- Show how to do it

Exercise 3: Making changes

- Follow the instructions in the exercise
- Make changes to your file
- Stage it
- Commit it
- Push it to GitHub



Why is Git Designed This Way?

Git and GitHub for Writers

The motivation behind the four stages

SDK BRIDGE

When would you make changes but not stage them?

- You don't stage changes until they are at a point where you expect to commit them
- You don't know yet that you are going to keep these changes
- Example:
  - Your project manager wants you to totally reorganize to see if it works
  - You may decide to just delete all the changes and go back to the old organization

When would you stage changes but not commit them?

- Staging gives you a final check to make sure it's what you want
- Example:
  - You stage some changes
  - You read over them one last time
  - If they are fine, you commit them
  - If not, then you continue editing, unstaging the files

When would you commit changes but not push them?

- Until they are pushed, they are private
- You might not be ready to share
- Example:
  - You finish a rough draft and commit it
  - You reread everything and decide to make some edits
  - You stage and commit the edits
  - Now you are ready and you push both commits

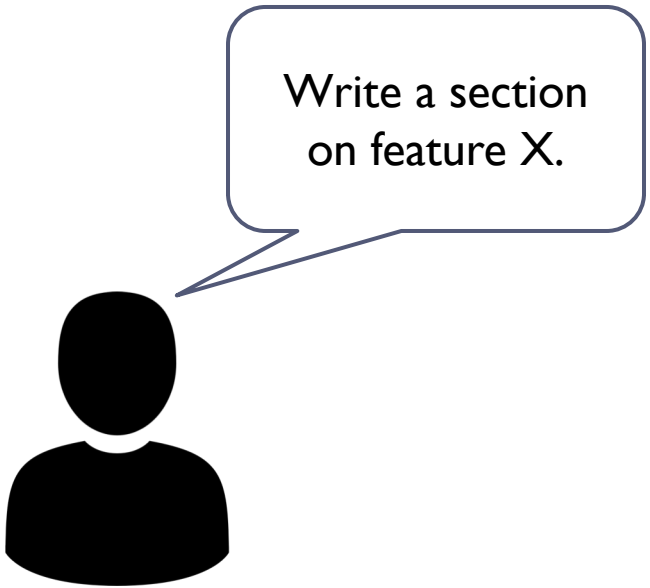
Going Back In Time

Git and GitHub for Writers

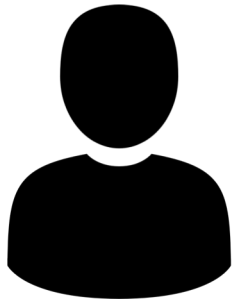
Working with previous commits

SDK BRIDGE

Without version control

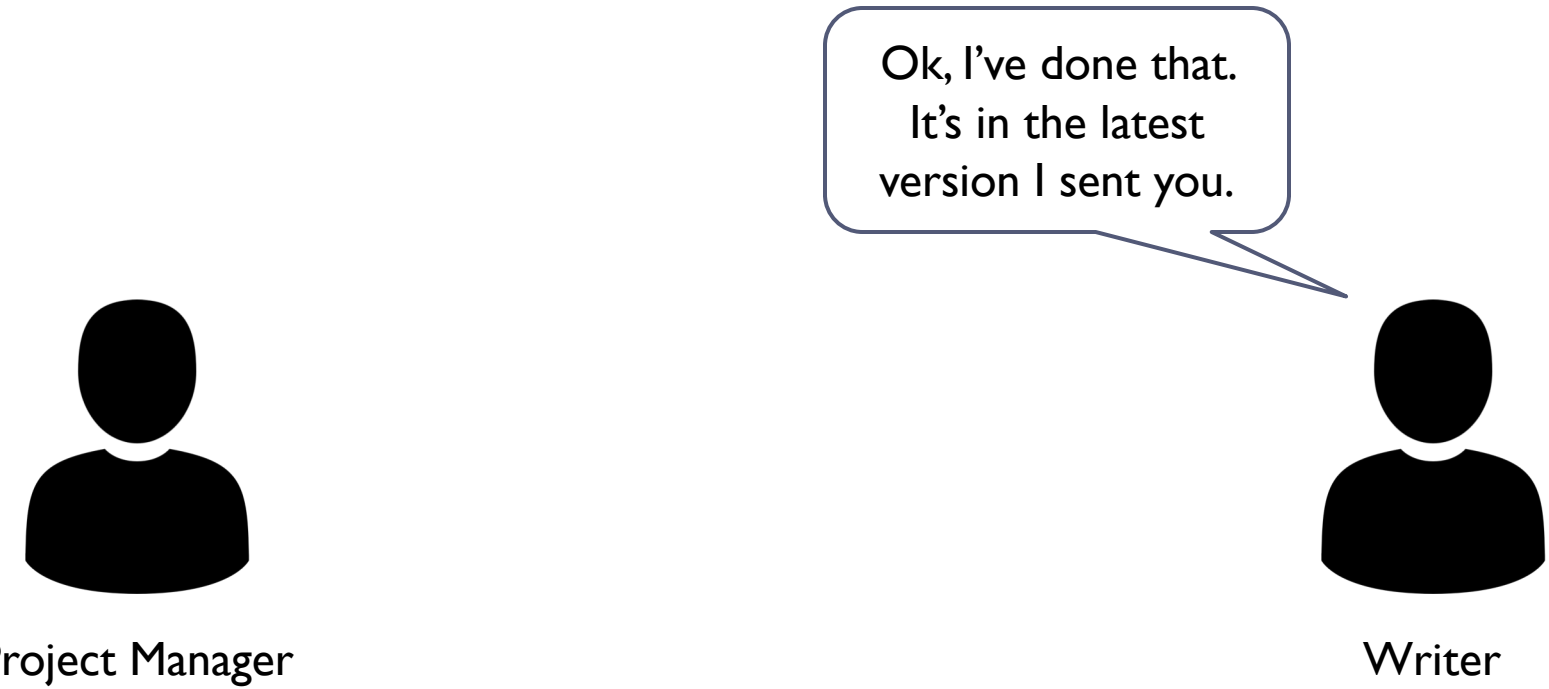


Project Manager



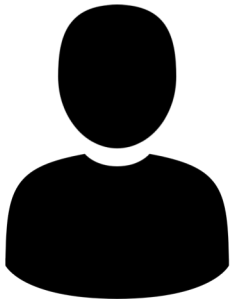
Writer

Without version control

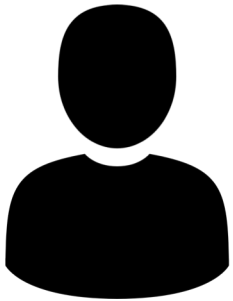


Without version control

One week later…

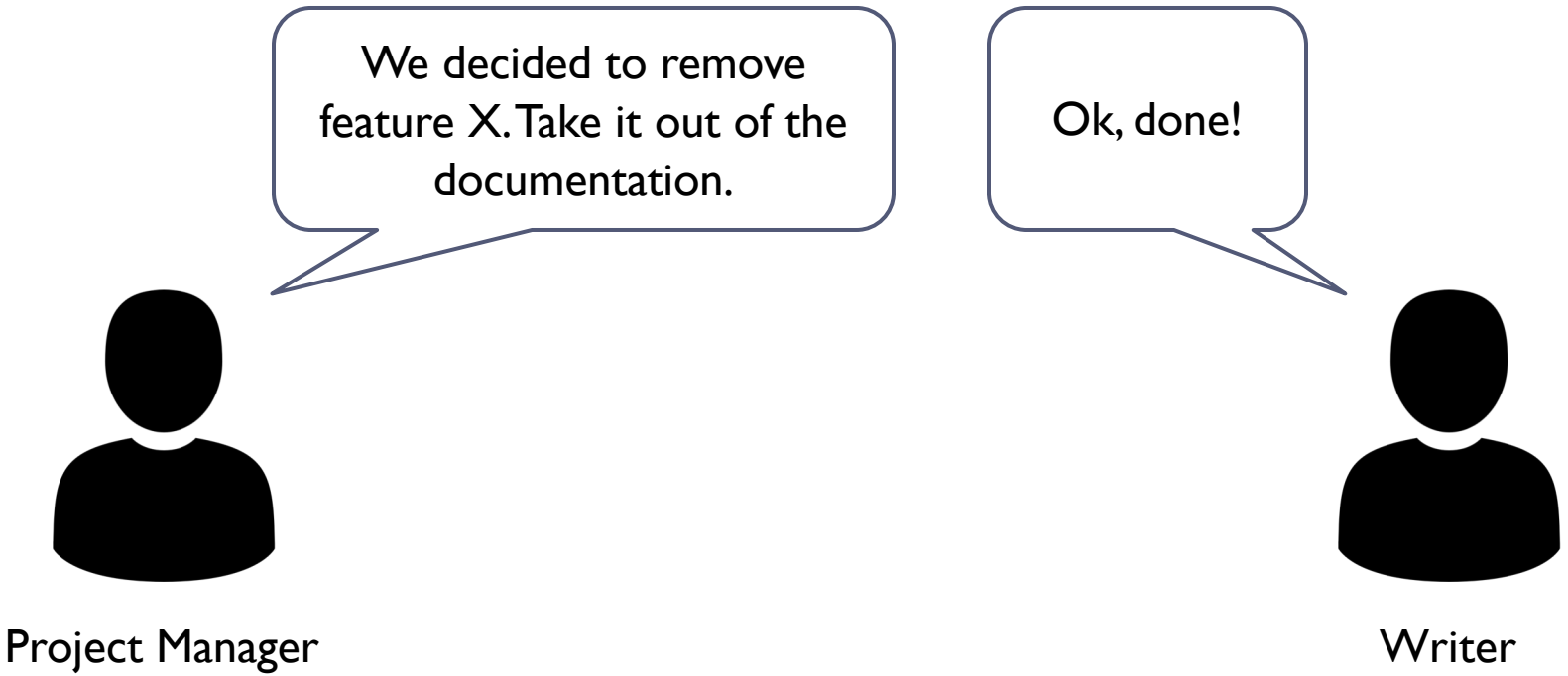


Project Manager



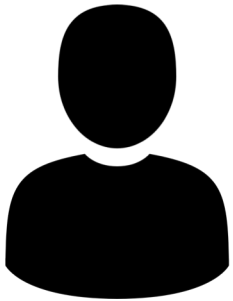
Writer

Without version control

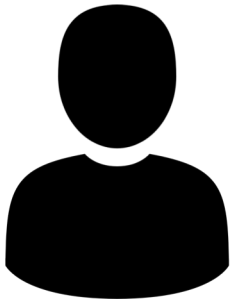


Without version control

Two weeks later…

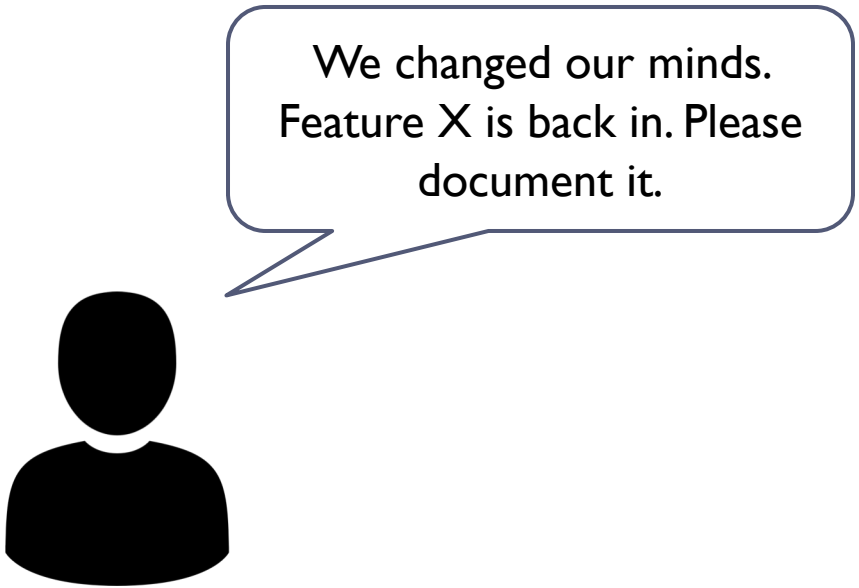


Project Manager

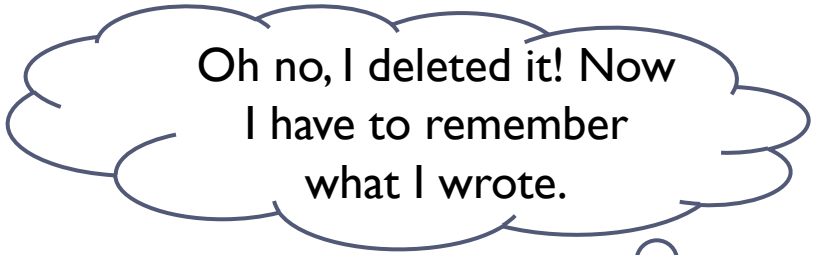


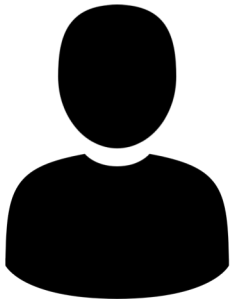
Writer

Without version control

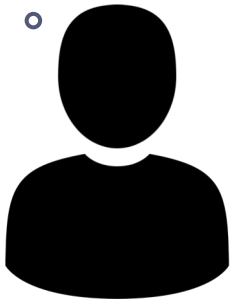


Without version control



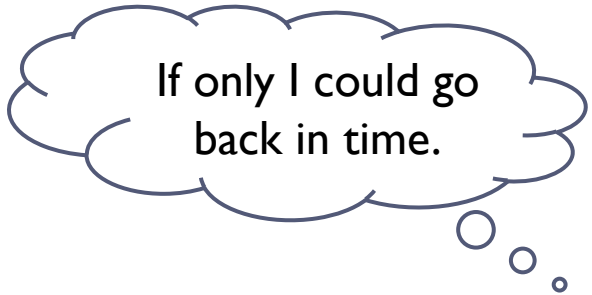


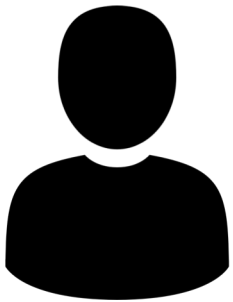
Project Manager



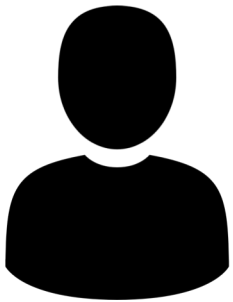
Writer

Without version control





Project Manager



Writer

GitHub

- GitHub can show you previous commits through the UI
- This allows you to easily look back and see what you did previously

HEAD

- Git labels a special commit as HEAD (all capitals)
- It means that commit you are currently working on
- This is usually the most recent commit you've made
  - Technically, the most recent you've made on a branch
- However, you can move HEAD to a different commit if that's something you want to work with

Git

- **`git log`** shows you your history
- **`git log --oneline`** presents a compact version
- **`git checkout`** is the a command that means:
  - Set the commit that I am working on
- Use **`git checkout`** to set the current version to an old commit
- Use **`git checkout master`** to set the current version to the most recent (on the master branch)
  - More on the master branch later

Placeholder

- Show everything in last slide
- Show how git uses hidden files

Best practices for commits

- Commit often
- Ideally each commit for changes contain just one change
  - For example: added a section, removed a section, changed the product name, etc.
- Clear, concise message for each commit
  - Lets you find changes easily

Exercise 4: Going back in time

- How to see commit history
- How to checkout old commit

Tags

Git and GitHub for Writers

Marking Special Commits

SDK BRIDGE

Tags

- Sometimes you want to label a commit as special
  - This is called a tag
- For example: you could tag a commit that corresponds to the files that are part of a release
- A tag is just a label applied to a commit to make it easier to find
- Tags have a name and a description
- Tags are applied locally. Use **`git push`** to upload it to GitHub

Git commands

- **`git tag -a`** adds a tag with name and description
  - `git tag -a my_tag -m "This is my new tag"`
- **`git tag`** lists all tags
- **`git checkout tags/<tagname>`** to checkout a tag
- **`git push origin <tagname>`** to push a tag

Pull

Git and GitHub for Writers

Downloading new versions

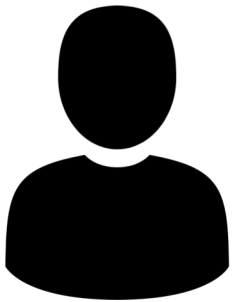
SDK BRIDGE

Collaboration

- The main advantages of version control:
  - Going back in time
  - Collaboration
- Version control lets multiple people work on a file at the same time
  - But you each have your own copy
- You can control how that those separate changes are incorporated

Collaboration

Repository

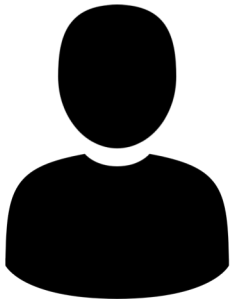


Developer

Writer

Collaboration

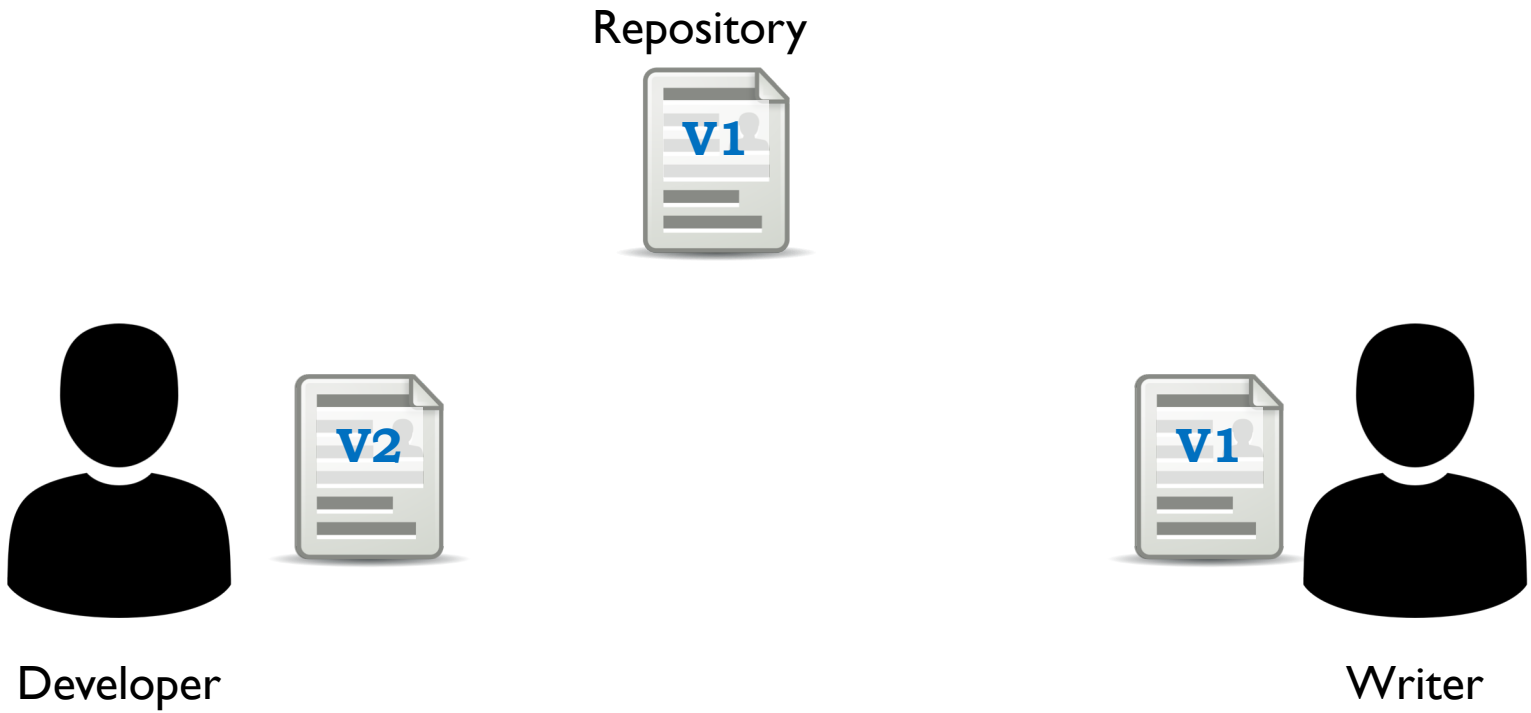
Repository



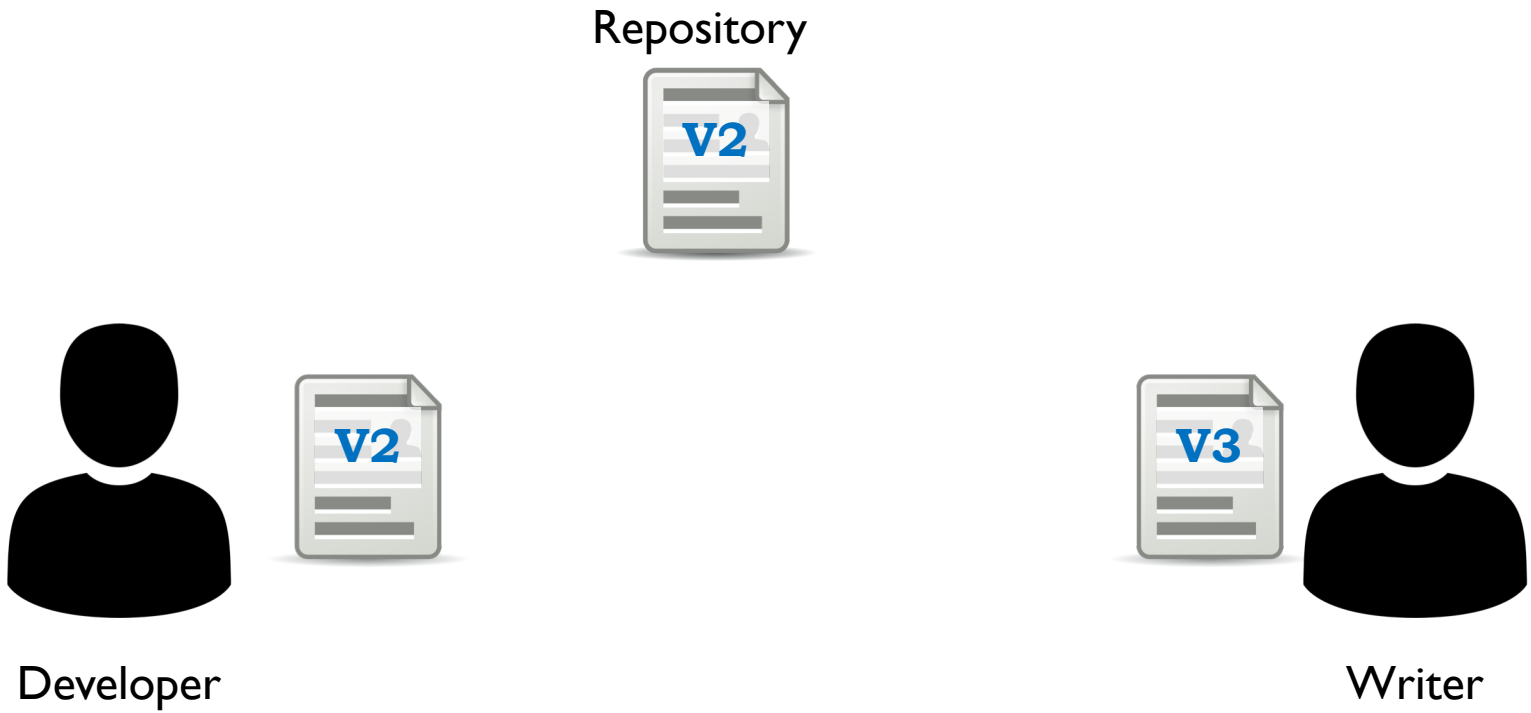
Developer

Writer

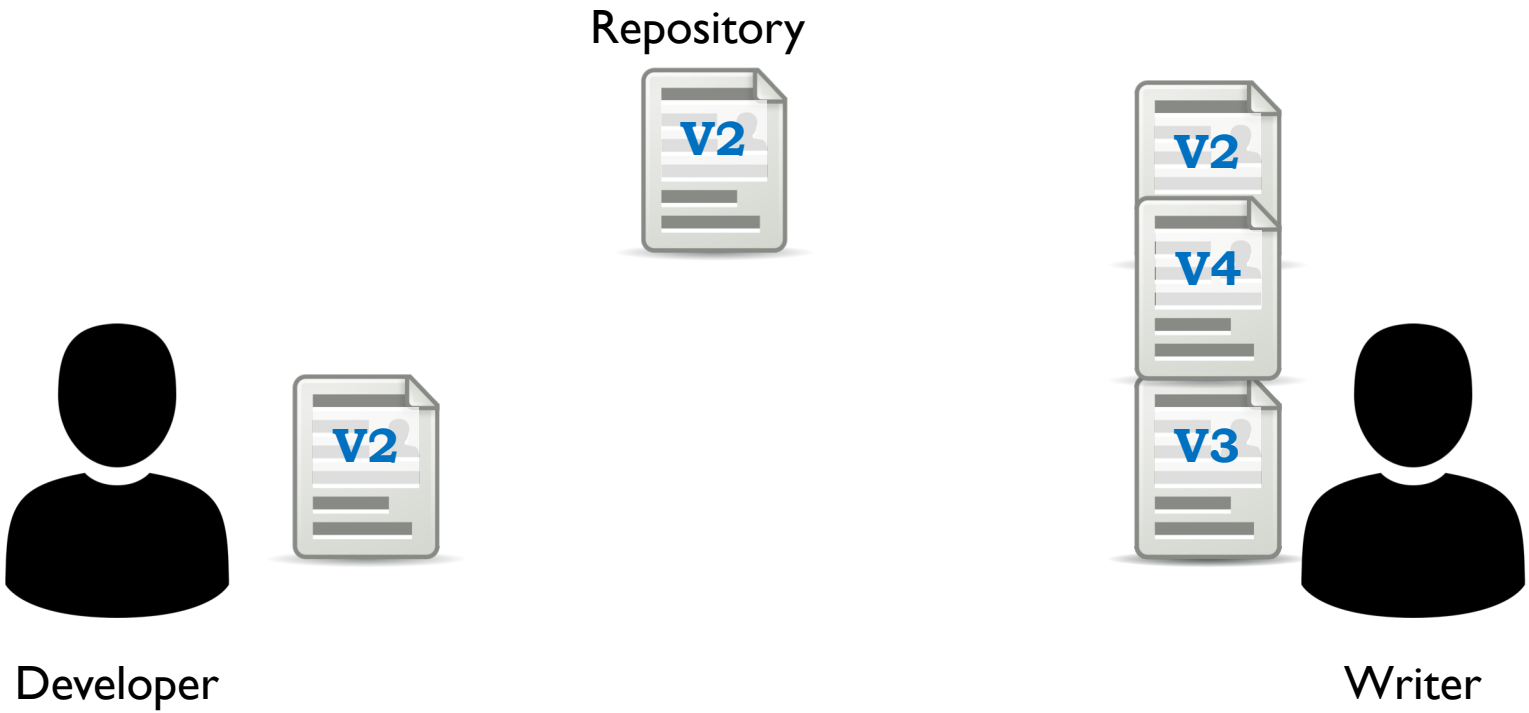
Collaboration



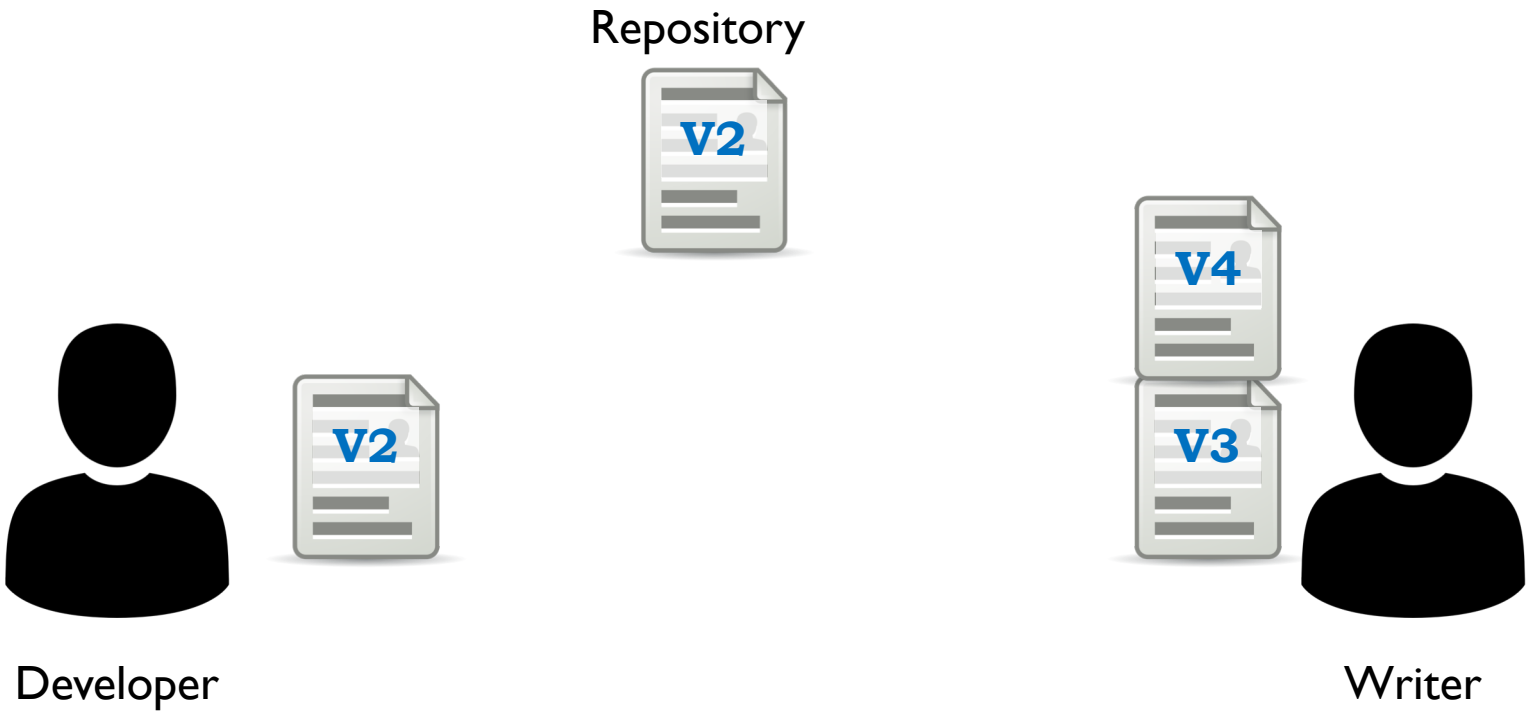
Collaboration



Collaboration



Collaboration



Pull

- **`git pull`** downloads all commits that have been pushed to the repository
- You can only do a **`git pull`** if you have committed your files
  - In other words, no merging unless you are at a good stopping place

Handling changes

- How does `git pull` handle changes? Three cases:
- Remote file hasn't changed since last pull you've done.
  - Leaves local file as is, whether you've changed it or not.
- Remote file has changed. Local file hasn't changed.
  - Replaces local file with remote file
- Both remote and local files have changed.
  - Remote file is merged into local file

Merging

- How does a merge happen?
- Git compares line-by-line, noticing which lines have changes
- If the line hasn't changed remotely or locally, then the line stays the same
- If a line has changed, either remotely or locally, it uses the changed line
- If the line has changed both remotely and locally, then this is called a "conflict"

Conflict

- If a line is changed both remotely and locally, then Git doesn't have enough information to know which one is right.
- It will notify you of a conflict
- It puts both the old and new ones in the file, marking them with <<<<<<< and >>>>>>> symbols, so you know where they are
- Edit the files, choosing how you want to merge and removing the <<<<<<< and >>>>>>> symbols
- Tell Git that the conflict is resolved

Placeholder

- Show how to do it

Binary files

- Git can only merge text files
  - Remember, it goes line by line
- Binary files (images, audio files, video files, etc.) cannot be merged
- You will need to choose a version
- If you need to merge, you will have to do it manually

Fetch

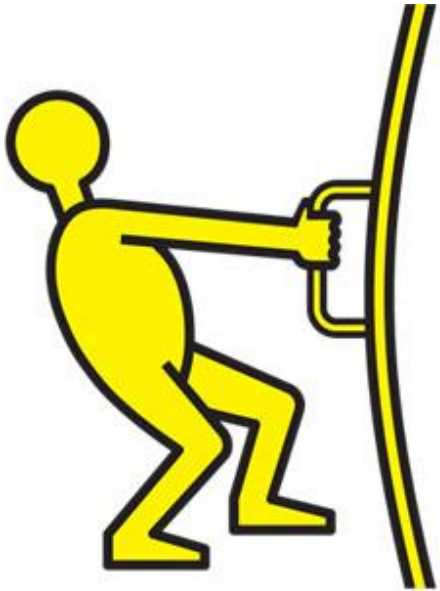
- You may come across the **`git fetch`** command
- This will download the remote files, but not merge them
- You will need to do the **`git merge`** command to merge
- **`git pull`** does both automatically. **`git fetch`** and **`git merge`** give you finer control.
- As a writer, you will probably only need to use **`git pull`**

Exercise

- Create a change on GitHub
- Pull the change and see it locally
- Create a change on both GitHub and locally
- Pull the change and merge
- Create a conflict
- Pull the change and resolve the conflict

Exercise 5: Pulling

- Make a simple pull
- Handle a merge
- Handle a conflict



Branches

Git and GitHub for Writers

Creating variations

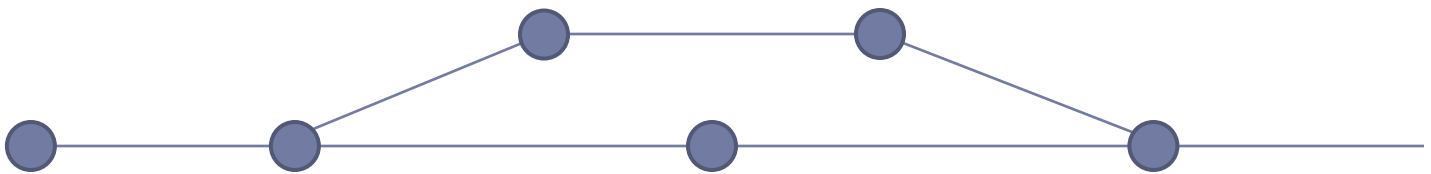
SDK BRIDGE

Branches

- Often it's useful to create a set of versions for a particular purpose
- You might create a new branch:
  - To add a new feature
  - To fix a particular bug
  - For documentation

Branches, continued

- Create a new branch when you start making changes for that particular purpose
- Once you have everything in the branch the way you want it, merge it back into the branch it came from



The master branch

- When a repository is created it has one branch called master
- The master branch is the root branch
- Unless you are on a very small project, you typically don't work directly on the master branch
- Instead, you create a branch, make commits, and then merge back into master
- You can branch off a branch that's branched off master

Creating a branch

- **`git branch` *`branch-name`*** creates a branch named *branch-name*
  - Branches off the current branch
  - But you stay on the current branch
- **`git checkout -b` *`branch-name`*** creates a branch and also moves you to that branch
  - Handy shortcut

Branch names

- Branch names can use letters, numbers, dashes, periods, and forward slashes
- They are case sensitive
- Most common is to use lowercase with dashes between words
  - Example: `api-docs`
- Slashes are used to create groups of branches
  - Example: `v1.2/api-docs, v1.2/login-fix`

Moving to a branch

- **`git checkout` *`branch-name`*** moves to the branch named *branch-name*
- You can only move to another branch if all files are committed
  - In other words, you want to be at a good stopping place
  - If you aren't ready to commit, you can use **`git stash`** (more later)
- **`git branch`** displays all branches and which one you are on

Deleting and pushing

- **`git branch -d` *`branch-name`*** deletes the branch named *branch-name*
  - Only do this once you have merged your changes or else you are sure you don't want that branch
- **`git push`** uploads everything to the server, including new branches
  - Until you do this, all new branches are local only
- Once pushed, you can delete the branch on GitHub
  - You will see the changes when do a **`git pull`**

Pull request

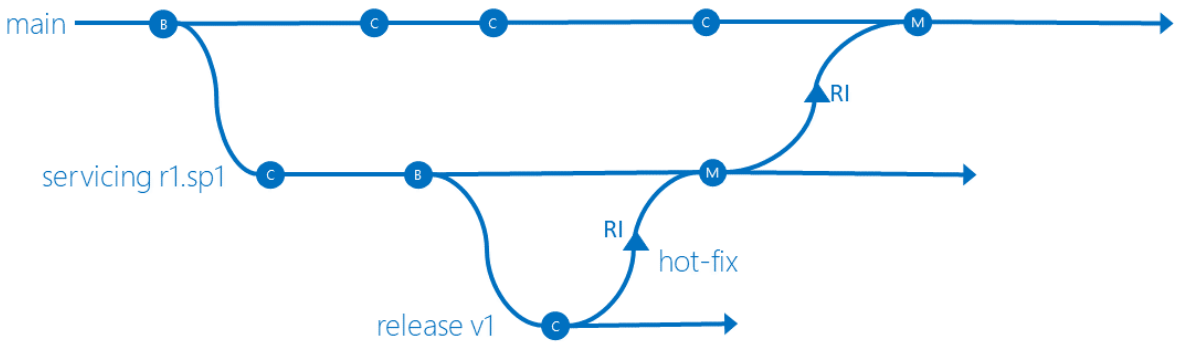
- Typically, at some point you want to merge your changes from the branch back into the branch they came from
- You can do this with the Git command line
- But it's easier to do with GitHub
  - This is called "Pull request" when done through GitHub
  - The idea is that you request your branch be merged and someone else can review and approve the changes.

Placeholder

- Show how to do it

Exercise 6: Branching

- Create a branch
- Create a pull request



Next Steps

Git and GitHub for Writers

Resources

SDK BRIDGE

What do you need to know next?

- You've learned a lot
- There are other concepts I haven't covered
  - Stash
  - Rebase
  - Forking

Resources

- I'm creating a Udemy course in the near future
- GitBook – a platform for documentation that uses Git.
- Docs Like Code by Anne Gentle

Other Udemy courses

- SDK Bridge Online courses
  - sdkbridge.com/online-courses
  - Workbook contains links with discounts