GETTING STARTED WITH GIT AND GITHUB

How I learned to stop worrying and love version control

David Chin, Drexel URCF

MOTIVATION

- Does you source code directory/folder look like this?
  - `my_program-4Feb2022.py`
  - `my_program-14Mar2022.py`
  - `my_program.py`
  - `my_program-algorithm1.py`
  - `my_program-algorithm2.py`
- Sidebar
  - If you want to use dates, use `YYYYMMDD` or `YYYY-MM-DD` which can be sorted numerically, e.g. `2022-02-04`, `2022-03-14`
- Fine for a very small number of files
- Does not scale
- Not sustainable, i.e. when you come back to your code some months/years later
- What is an effective an efficient way of keeping track of changes?

WHAT YOU WILL LEARN

- By the end of this talk, you should be able to:
  - Explain why version control is useful
  - Create a new GitHub repository
  - Use a simple (branchless) Git workflow
    - Edit
    - Add (Stage)
    - Commit
    - Tag
    - Push
  - Use a simple branching Git workflow
    - Create branch
    - Switch branch
    - Merge changes
  - Recover a previous state of code

VERSION CONTROL

- a.k.a. revision control, source control, or source code management (SCM)
- Class of systems responsible for managing changes to computer programs, documents, large web sites, or other collections of information
- Two major classes
  - Centralized
    - One central repository holds the “truth”
    - Only one person modifies one part of the code at a time
  - Distributed
    - All repositories are equal
    - Repositories can be synced to each other

WHAT IT CAN DO FOR YOU

- Allows for experimenting with new sections of code while enabling reversion to older known working state
- Allows for collaboration with careful rules about “clobbering” (overwriting) each other’s work
- Allows meaningful version numbers
- Not just for computer code: I used version control on my dissertation

GIT

- Originally written by Linus Torvalds (author of Linux) in 2005
- Used to manage Linux kernel source code: ~25 million of lines of code, thousands of developers, all making modifications at the same time
- Can be complex but only small subset of commands needed for useful work

GITHUB

- While git is distributed, it is helpful to have a conceptually central repository
  - For a project with multiple developers in different locations, their PCs may not be able to communicated directly with each other to sync changes. GitHub serves as an intermediary.
- Students get some “pro” features for free
- Provides own GitHub CLI tool called “gh”
  - We will not use it here

ALTERNATIVES TO GITHUB

All these work with Git:

- GitLab
- BitBucket
- GitBucket
- AWS CodeCommit
- SourceForge
- Google Cloud Source Repositories
- Phabricator
- Gitea (self-hosted)
- Apache Allura
- Launchpad (by Canonical, distributors of Ubuntu Linux)
- Ref: https://www.geeksforgeeks.org/top-10-github-alternatives-that-you-can-consider

SIGN UP FOR A GITHUB ACCOUNT

https://education.github.com/discount_requests/student_application

SSH KEYS

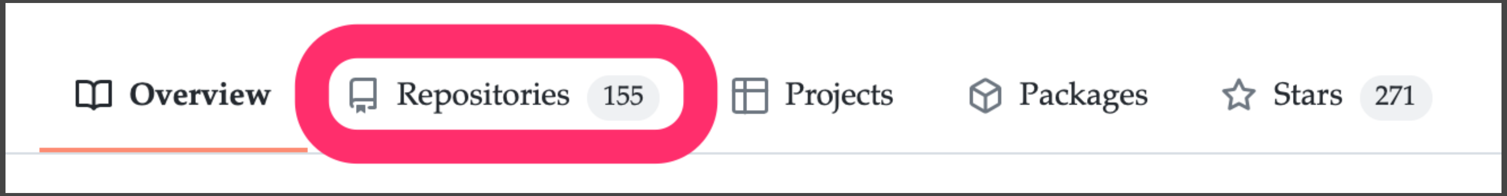
https://docs.github.com/en/authentication/connecting-to-github-with-ssh

SSH KEYS

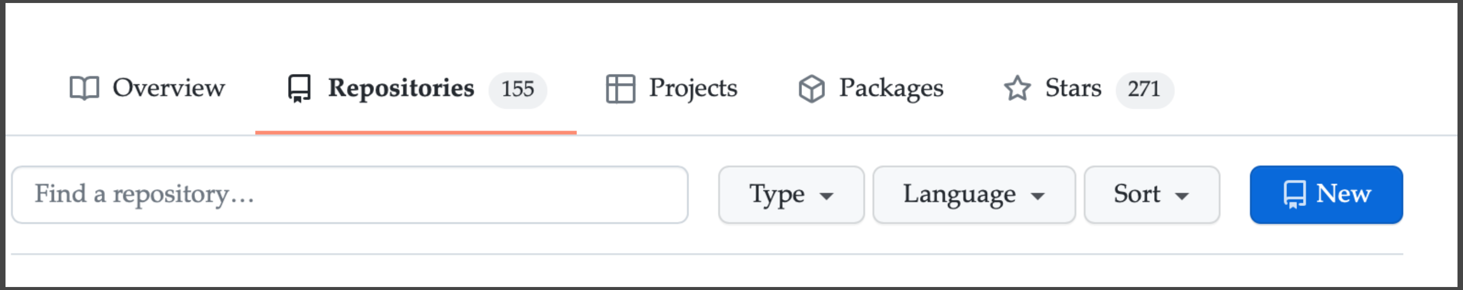
- Allows passwordless connection with GitHub
- Generate key:
  - `ssh-keygen -t ed25519 -C your_email@example.com`
  - DO NOT USE AN EMPTY PASSPHRASE
- Start the SSH agent:
  - `eval "$(ssh-agent -s)"`
- Add your key to the agent:
  - `ssh-add ~/.ssh/id_ed25519`
- Add the **public key** to your GitHub account:
  - `cat ~/.ssh/id_ed25519.pub`
- Test connection to GitHub:
  - `ssh -T git@github.com`
- Ref: https://docs.github.com/en/authentication/connecting-to-github-with-ssh

CREATE A NEW REPO

- "repo" = repository
- Go to your repositories page:
  - `https://github.com/yourname`



- Click the "New" button



IMPORT EXISTING CODE TO GITHUB

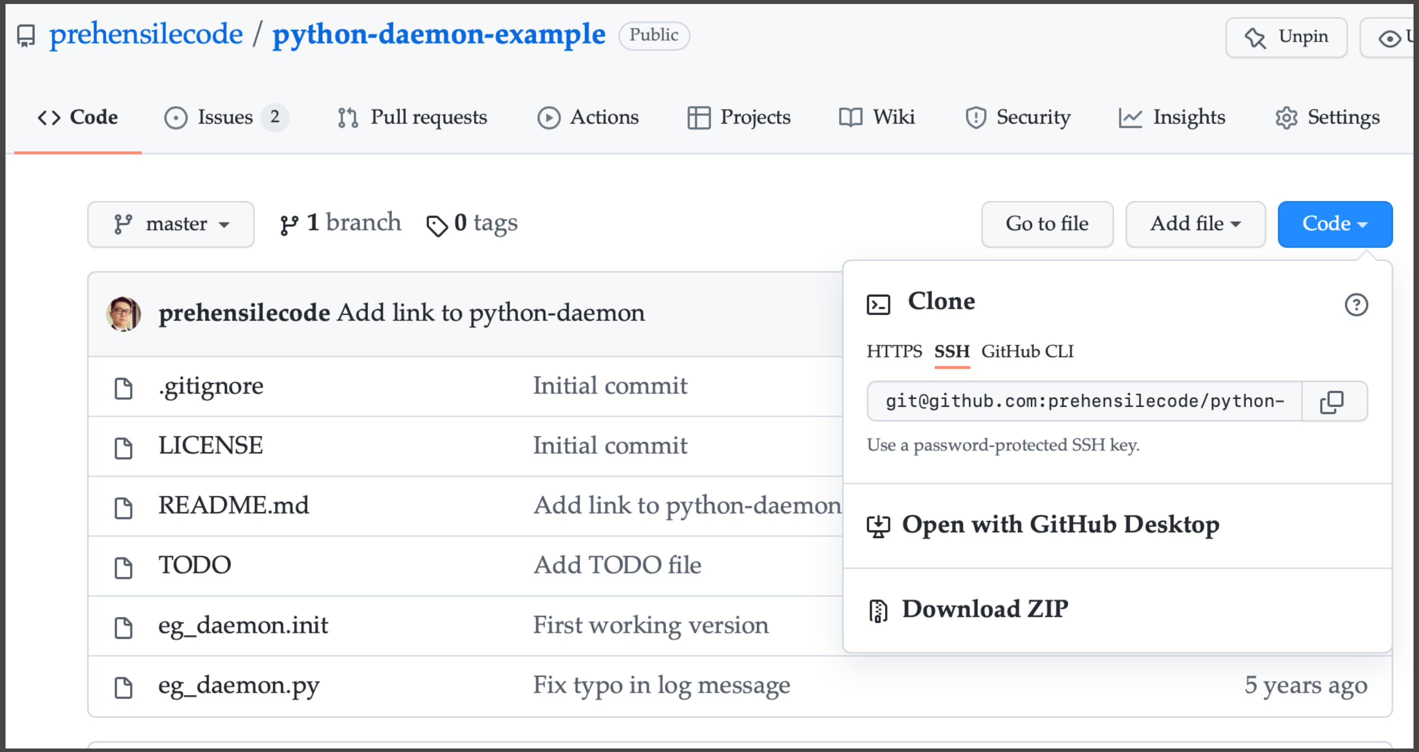
- Create a new repo at GitHub
- On your PC, move existing code directory to a different name, e.g. myproject_orig
- Clone the repo to your PC
- Copy all the original code to the cloned repo
- Add/Stage, commit, and push all the original files to GitHub
- Optionally, save your “myproject_orig” to an archive location

GIT CONFIGURATION

- Show current configuration
  - `git config --global --list`
- Modify some configurations:
  - `git config --global user.name "Sam Noone"`
  - `git config --global user.email abc123@drexel.edu`
  - `git config --global core.editor nano`
    - Alternatively, change your environment variable `VISUAL` and/or `EDITOR` to "`nano`" or whatever editor you prefer
  - `git config --global init.defaultbranch main`

CLONE THE REPO

- Copy the "ssh" repo link
- In terminal:
  - `git clone git@github.com:myname/myrepo`



prehensilecode / python-daemon-example Public

prehensilecode Add link to python-daemon

.gitignore	Initial commit	
LICENSE	Initial commit	
README.md	Add link to python-daemon	
TODO	Add TODO file	
eg_daemon.init	First working version	
eg_daemon.py	Fix typo in log message	5 years ago

Clone

HTTPS SSH GitHub CLI

`git@github.com:prehensilecode/python-`

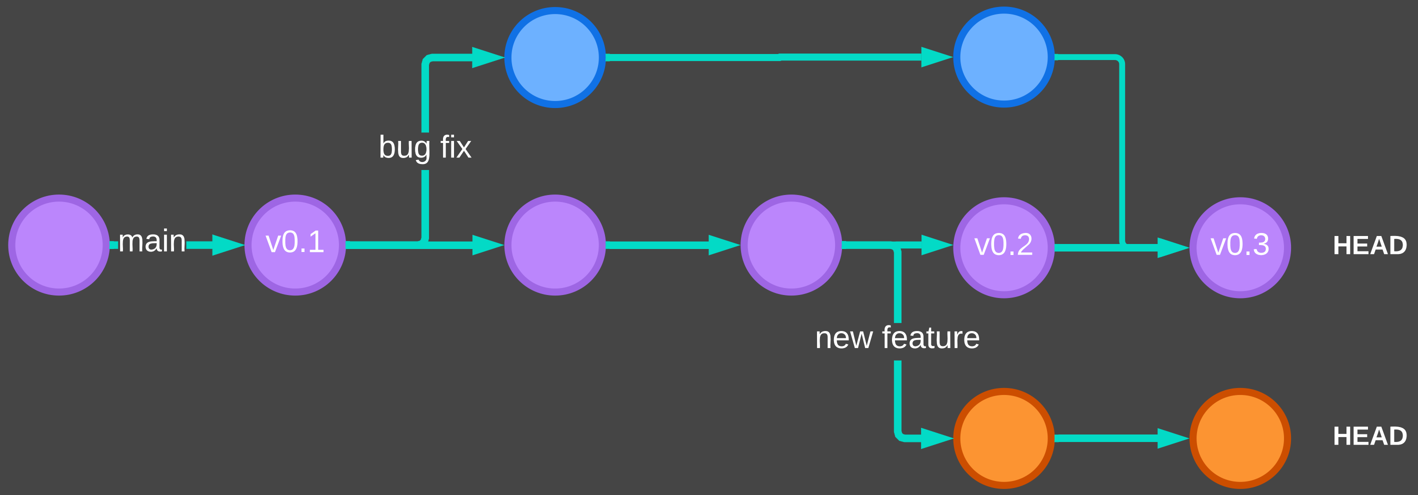
Use a password-protected SSH key.

Open with GitHub Desktop

Download ZIP

DIAGRAM OF STATE OF CODE

- Each circle represents a "commit"



CREATE A NEW FILE

- Type (or use editor of choice):
  - `nano hello.py`
- Edit
- Save

Contents of file:

```
#!/usr/bin/env python3
print("hello, world!")
```

ADD/STAGE THE FILE

- Type:
  - `git add hello.py`

COMMIT THE CHANGE

- Type:
  - `git commit`
- An editor will launch asking for a commit message
  - Type a brief description of the changes you made
  - Save the commit message (file) and quit the editor
- N.B. you are committing this change to your **local** repository
- Check the repo on GitHub
  - Notice that the file is not there
- DEMO

PUSH THE CHANGE UPSTREAM

- Type
  - `git push –u`
- Check the repo on GitHub
  - Notice that the file is now there, with the commit message shown
- DEMO

MODIFY AN EXISTING FILE

- Now, edit your file and make some changes
- Save the file
- See a summary of what you have changed:
  - `git diff`
- Then: `add, commit, push -u`
- DEMO

SUMMARY SO FAR

1. Add/Stage file(s) – `git add`
2. Commit file(s) – `git commit`
3. Push upstream – `git push -u`
4. Make changes
5. Go to 1

Aside: to make shell have git decorations https://ohmybash.nntoan.com

UNDO

- Undo an edit which has not been staged/added
  - `git restore .`
  - `git restore path/to/file`

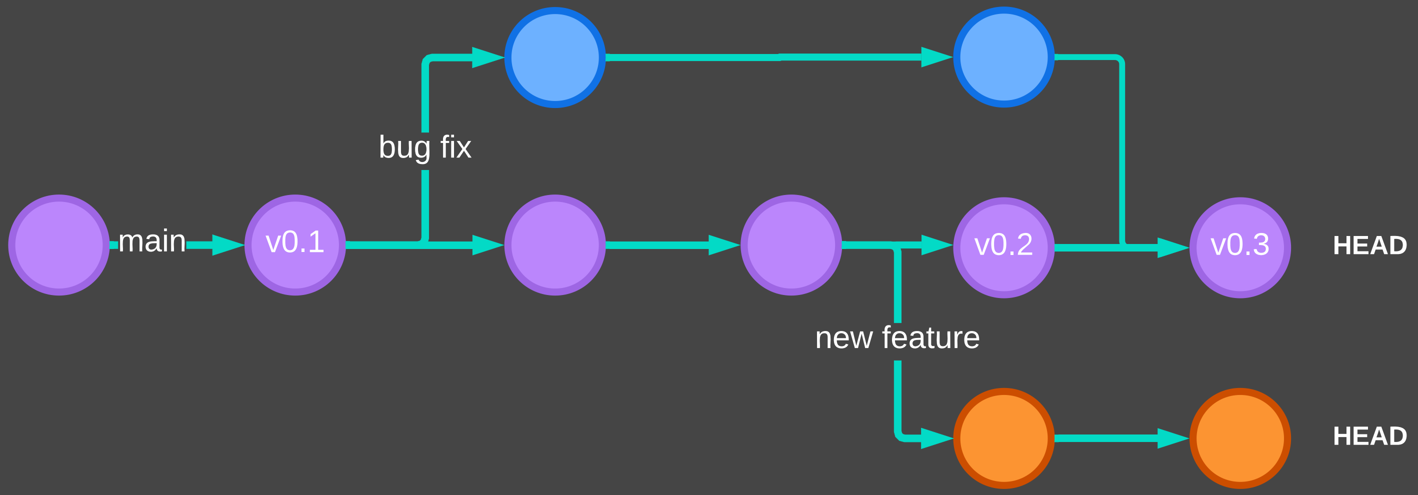
UNDO A COMMIT

- Oops. How to roll back a bad commit?
  - **Before** it has been pushed upstream
- Reset to one commit before HEAD:
  - Retain changes: `git reset --soft HEAD~1`
  - Discard changes: `git reset --hard HEAD~1`
- Hazard of only working with a single branch
- DEMO

BASIC BRANCH AND MERGE

- How to make modular changes to code without breaking what is already working
  - Including fixing bugs
- How to collaborate without stepping on each other's toes
- How to get back to a previous working version
- You do not need branch and merge for a very basic workflow, e.g., https://uidaholib.github.io/get-git/3workflow.html
  - BUT very useful for "unbreaking" things
  - Especially if multiple people are working on the same code
- More docs: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

BASIC BRANCH AND MERGE

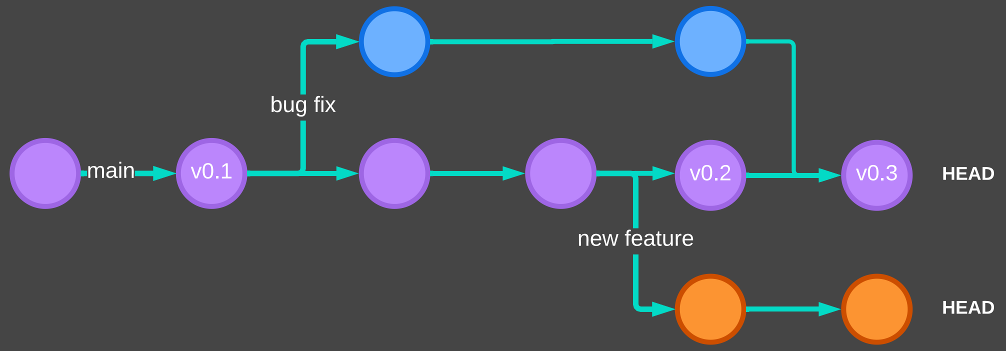


BASIC BRANCH AND MERGE

- DEMO
  - Create a new branch and switch to it
  - Work on new branch
  - Compare with `main` branch
  - Commit new branch
  - Merge new branch into `main`

BASIC BRANCH AND MERGE DEMO

- `git checkout -b new-feature`
- List all branches
  - `git branch`
- Edit new file `goodbye.py`
- `git add goodbye.py`
- `git commit`
- `git push –u`
  - Read error message, and follow directions
- `git push --set-upstream origin new-feature`
- Go to GitHub
  - Read the message



BASIC BRANCH AND MERGE DEMO

- Merge the change into `main`
  - `git checkout main`
  - `git pull origin main`
  - `git merge new-feature`
  - `git push origin main`
- Delete the "new-feature" branch
  - `Use GitHub on web`
  - Command line
    - `git branch –d new-feature`
    - `git branch` (to check)
    - Push the change (branch delete) to GitHub
    - `git push --delete origin new-feature`
  - Check on GitHub

SUMMARY SO FAR

- To fix a bug, or add a new feature
- Create a new branch and check it out (a.k.a. switch to the new branch)
- Make edits in the new branch and commit as usual
- Once satisfied (i.e. bug fixed, or feature fully implemented)
    - Merge branch back into `main`
    - Optionally, delete the bug fix/feature branch

TAGGING

- For major or minor "releases"
- Known working versions
- Ref: https://git-scm.com/book/en/v2/Git-Basics-Tagging

TAGGING DEMO

- We now have first working version of our application
- Create an annotated tag
  - `git tag –a v0.1 -m "First working version 0.1"`
- See all tags
  - git tag
- Show annotations
  - `git show v0.1`
- Push tags to GitHub
  - `git push origin –tags`
- Look on GitHub
  - N.B. you can download a zip or tar.gz archive file
- Branch, add new feature, merge, tag
  - Use gist https://gist.github.com/prehensilecode/c18eeb5876c6c8b64ec681ad691e6910
- Switch to old tag
  - `git checkout v0.1`

SUMMARY SO FAR

- Tag to mark significant milestones, e.g. releases
  - Tags should be applied only to fully working code (barring any undiscovered bugs)
- Tags allow you to “rewind” to a previously working state
  - If bugs are too major to just edit to fix, you can discard any changes made since a previous tag. E.g. v1.3 is badly broken, rewind to v1.2 by:
    - `git checkout tags/v1.2`

WHAT GITHUB ADDS TO GIT

- Not just a remote Git repository
- Issue tracking
  - Bugs
  - Feature requests
- Collaboration
  - Pull requests – others (may or may not be in team) can contribute code, and request that the owner of the repo pull the change into the originating repo
- *Continuous Integration* (CI)
  - is the practice of merging all developers' working copies to a shared mainline several times a day
  - Use GitHub Actions:
    - https://docs.github.com/en/actions
    - https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions
    - e.g. run a test suite (on GitHub servers) whenever a change is pushed

GITHUB CLIENTS

- GitHub Desktop – GUI Application
  - https://desktop.github.com/
  - Available for macOS and Windows only
- GitHub Command Line Interface
  - https://cli.github.com/
  - Access GitHub functionality from the command line (I.e. not just another git)

SUGGESTED WORKFLOW FOR GETTING CODE TO PICOTTE

- Do no (or minimal) editing on Picotte
- Create a GitHub repo (private or public)
- Edit code on your personal computer
- Push changes to GitHub
- Go to Picotte
- Checkout the repo, and pull any updates, and run code
- Rinse and repeat

SUMMARY

- Basic source code management with git
- Edit, stage (add), commit, push
- Pull
- Branch and merge
- Tag
- GitHub features
- Workflow suggestion
- Importing existing code to GitHub

EDITORS

- Code editors provide features to aid in programming
  - Syntax highlighting
  - Error checking
  - Etc.
- Extensible
  - e.g. Run a Jupyter notebook in a VS Code (or Emacs) tab
  - Support Git and GitHub

INTEGRATED DEVELOPMENT ENVIRONMENTS (IDES)

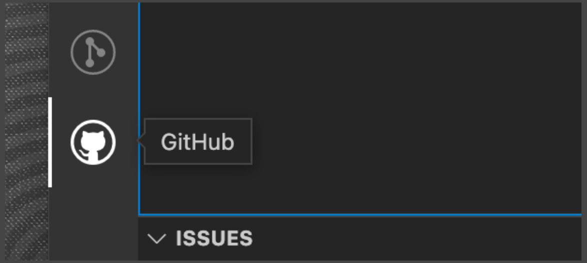
- UI to manage entire software projects
    - Build and test system
    - Debugger
    - Tracing
    - Version control
- Editors now have some IDE features (see previous slide)
- E.g.
    - Eclipse https://www.eclipse.org/ide/
    - NetBeans https://netbeans.apache.org/download/index.html
    - macOS XCode

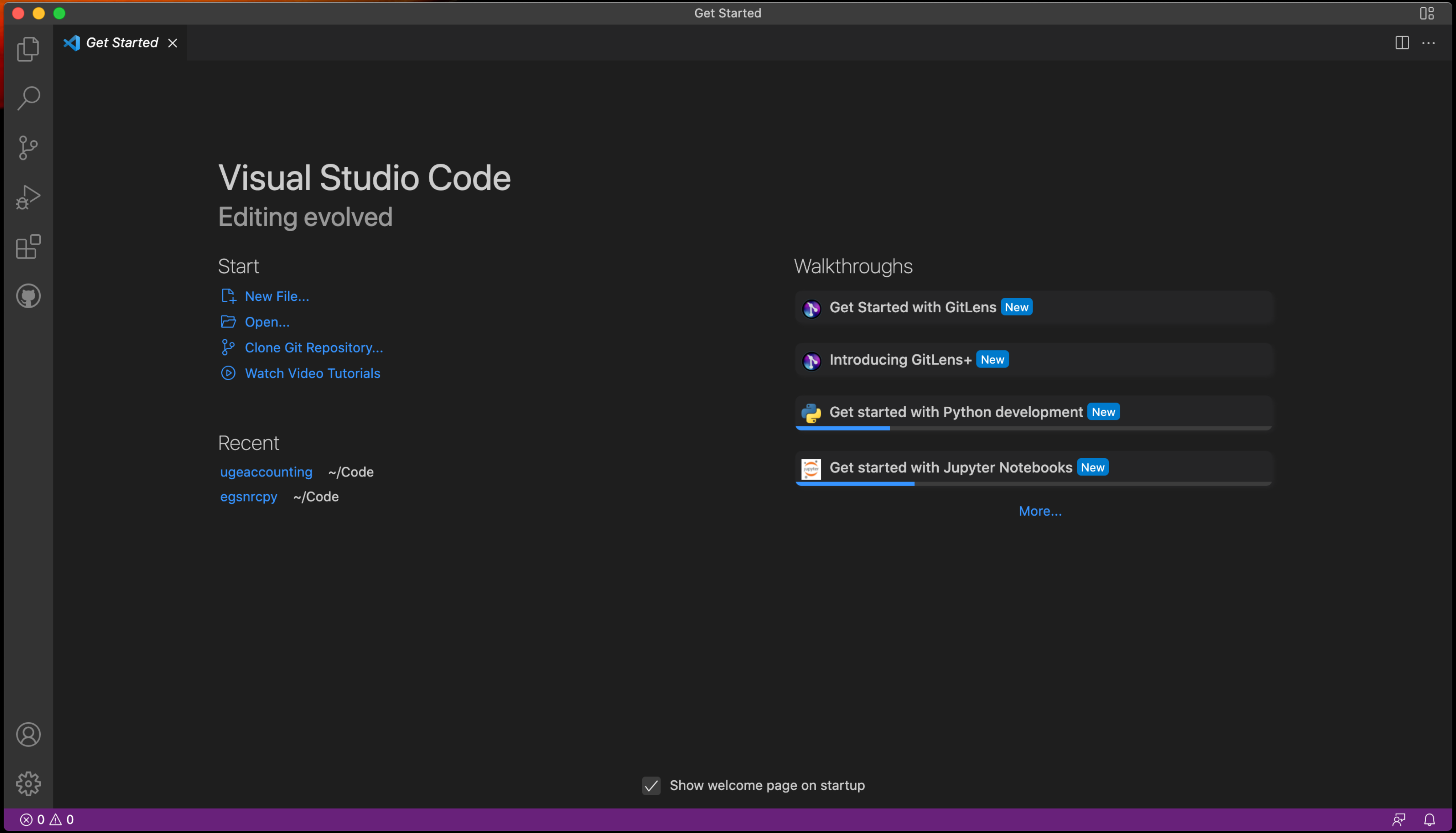
EDITOR INTEGRATION

- Using a separate Git/GitHub application or command line is a little annoying
  - Switch from editing to GH Desktop or terminal to perform Git operations
- Editors for programming almost all have integration with Git and GitHub
  - Indicators while editing to show state of code
  - Perform git operations: add/stage, commit, push, tag, etc.
  - Perform GitHub operations: refer to an issue, etc.

EDITOR INTEGRATION: VS CODE

- VisualStudio Code
  - Download: https://code.visualstudio.com/
  - Free of charge editor from Microsoft
  - Runs on Windows, macOS, and Linux
- Has Git and GitHub integration
  - Git (and other SCM) support built in
    - But there are other extensions which offer convenience functionality
    - See: https://code.visualstudio.com/docs/editor/versioncontrol
  - Extensions to install:
    - GitLens
    - GitHub Pull Requests and Issues
    - Etc.
  - Sign in to GitHub





Get Started

Visual Studio Code

Editing evolved

Start

New File...

Open...

Clone Git Repository...

Watch Video Tutorials

Recent

ugeaccounting ~/Code

egsnrcpy ~/Code

Walkthroughs

Get Started with GitLens New

Introducing GitLens+ New

Get started with Python development New

Get started with Jupyter Notebooks New

More...

✓ Show welcome page on startup

VS CODE WITH GIT DEMO

EDITOR INTEGRATION: EMACS

- Emacs is not just an editor
  - It is an entire user interface that can replace your desktop, including shell, email, web browser, calendar, ipython, etc.
  - Because it is a machine which runs apps written in the Emacs Lisp language
  - The editor is just the default app running
  - Runs both in GUI and in the terminal
- Emacs vs Vi(m) is archaic
  - Power users use Emacs with Spacemacs and Vi key bindings
- Spacemacs
  - No funny control sequences, only SPACE
  - Discoverable: possible completions shown every time so you can learn as you go
- GOAL: don't move hands from keyboard

WHAT IS VI

- `vi` is a modal editor
  - *insert mode* - where typed text becomes part of the document
  - *command mode* - where keystrokes are interpreted as commands that control the edit session
- Command mode allows powerful operations
  - Repetitions
  - Search and replace
  - Etc.

EMACS WITH GIT DEMO

REFERENCES

- Git book: https://git-scm.com/book/en/v2
  - About version control: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
  - Branch & merge: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging
- GitHub quick start: https://docs.github.com/en/get-started/quickstart
- Simple git workflows:
  - https://uidaholib.github.io/get-git/3workflow.html
  - https://www.atlassian.com/git/articles/simple-git-workflow-is-simple
- Another branch & merge tutorial: https://www.atlassian.com/git/tutorials/using-branches
- GitHub CLI (gh): https://cli.github.com
- Oh My Zsh: https://ohmyz.sh
- Oh My Bash: https://ohmybash.nntoan.com
- Visual Studio Code: https://code.visualstudio.com/
- Sublime Text: https://www.sublimetext.com/
- Spacemacs: https://www.spacemacs.org