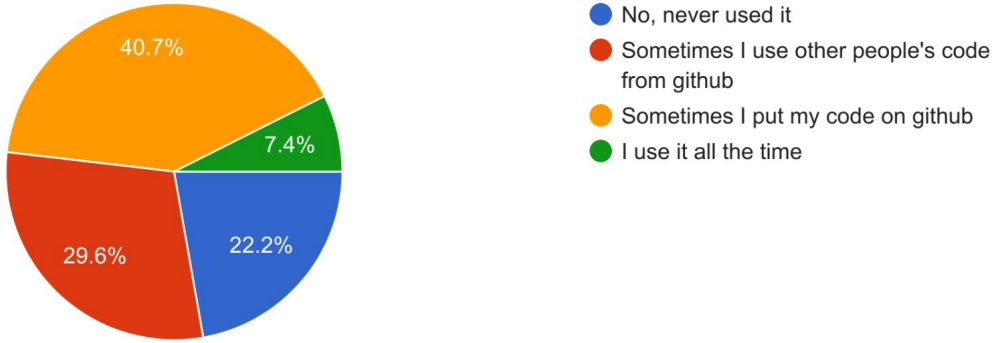
SMP git and github workshop

https://github.com/benroberts999/git-workshop

Ben Roberts
https://broberts.io/teaching

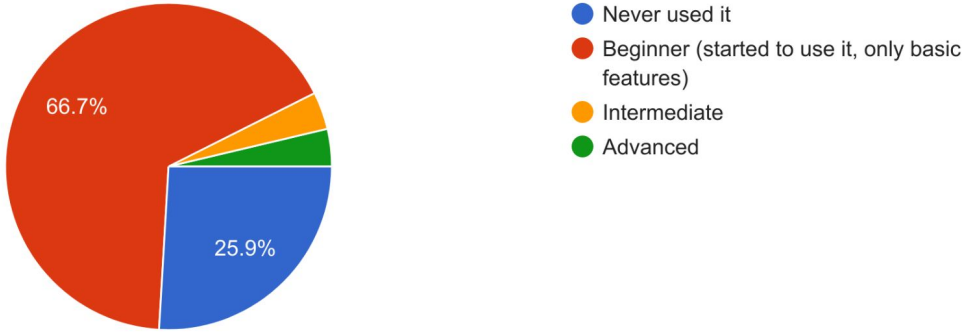
Have you used github.com (or gitlab or similar)?

27 responses



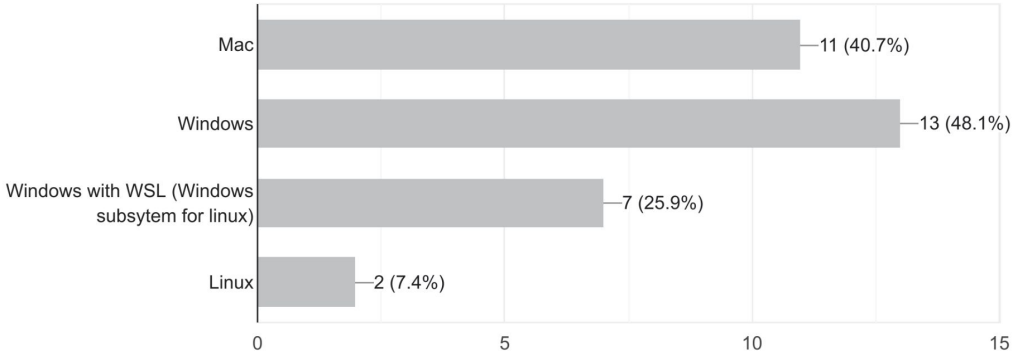
What's your current experience with git?

27 responses



What operating system do you use for coding?

27 responses



How comfortable are you with using the command line?

27 responses

xkcd.com/1597/

git

- **Version control**
  - Snapshots (entire project), called “commits”
  - Stores diff/delta only: great for plain text
  - Easily re-wind, backtrace errors
- **Branching**
  - Simultaneous development
  - Edit code without impacting working code
  - Merge into main branch when working
- **Remote synchronisation**
  - Share and store code remotely
  - Allows collaboration
  - Github: website to host projects “repositories”
- **Somewhat steep learning curve**
  - Ugly interface: can wrote learn, but
  - Basic understanding helps immensely
  - Interactive workshop

Git: basics

- Snapshots of entire project: commits
- Commit: what, who, why
- Graph: commits are nodes
- HEAD is current node
- Branch (give name to nodes)

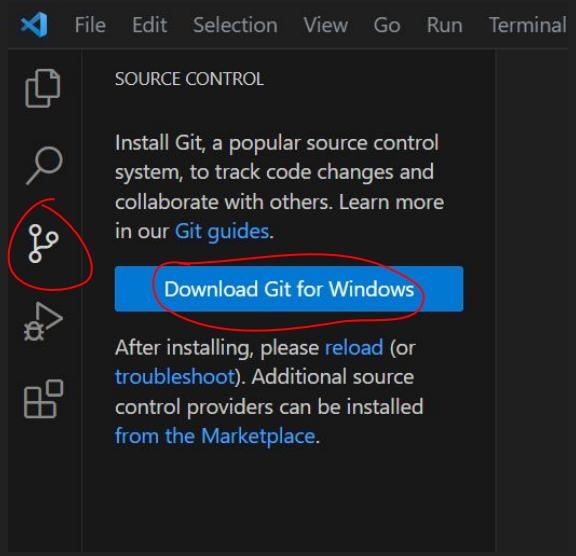
- Commit contains:
  - Data (form of a diff)
  - Author
  - Message
  - Parent(s) (preceding commits)
  - Id: hash of the data

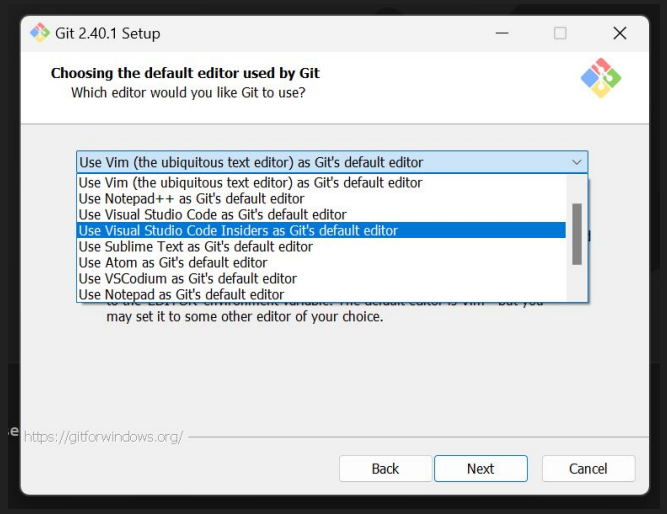
Workshop Tasks

- Head over to: https://github.com/benroberts999/git-workshop

- Pre-work
- Make yourself a github user account https://github.com
- Install Visual Studio Code: https://code.visualstudio.com/
  - Install git (next slide)
- (optional) If you want to use the command line, set up ssh keys: https://docs.github.com/en/authentication/connecting-to-github-with-ssh

Install git on windows (use vscode)





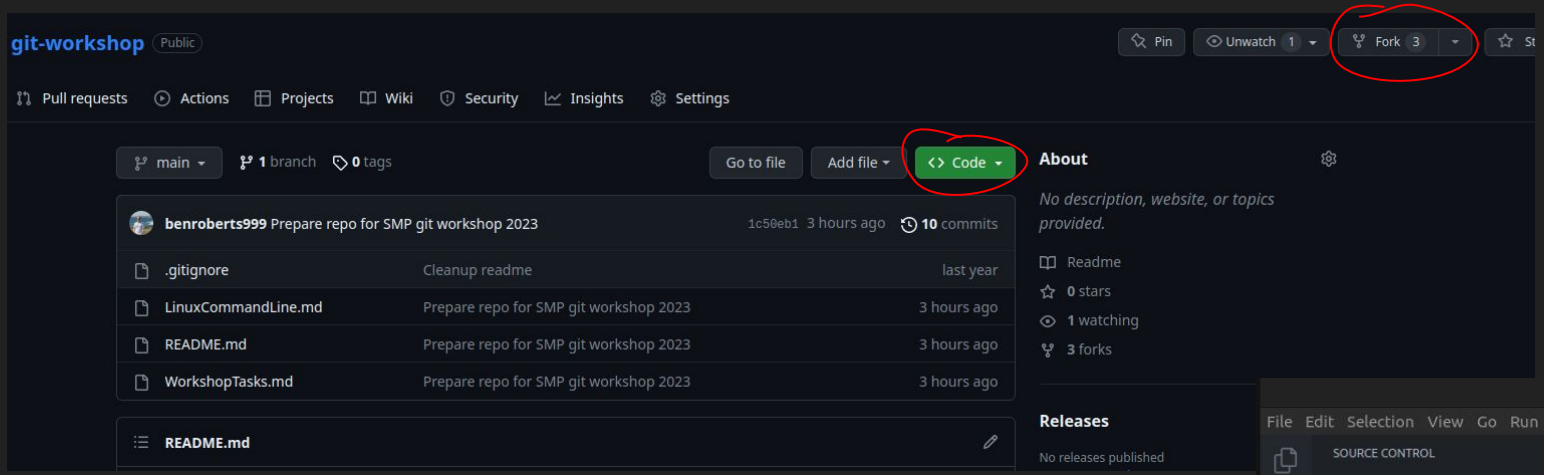
Install on mac: https://www.atlassian.com/git/tutorials/install-git

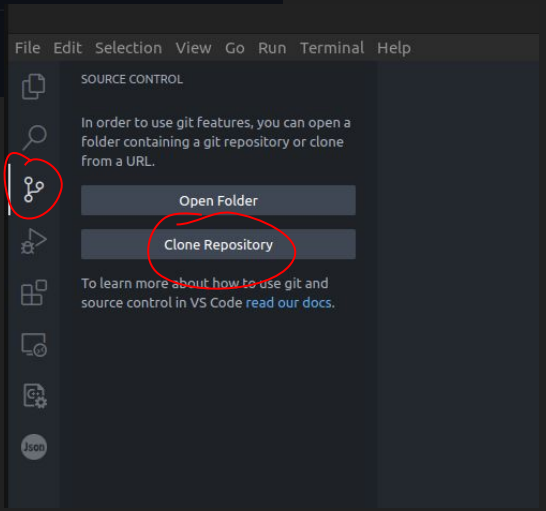
```
$ git config --global user.name "Emma Paris"
$ git config --global user.email "eparis@atlassian.com"
```

Summary of tasks

1. Fork + clone repo (or create new one)
2. Add some changes: commit
3. Push to github
4. Make changes on github, and pull them down
5. Create a new branch: make commits and swap between branches
6. Merge our branches

Fork and/or clone





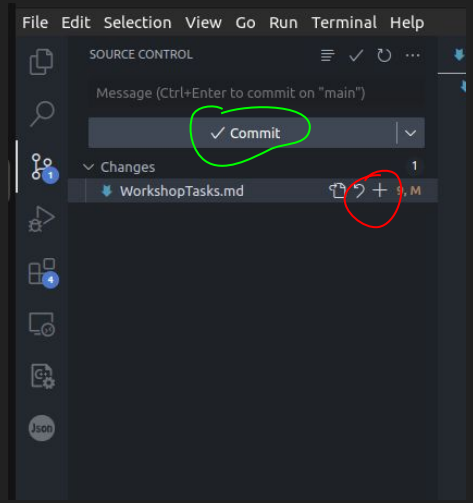
- Fork: copy repo to your github
- Clone: download git repo to your pc
- *git clone git@github.com:benroberts999/git-workshop.git*
- Or, GUI =>
- Or, initialise a new repo with ‘git init’
- DON’T put a git repo inside dropbox/onedrive etc.

Interlude: ssh keys

- To set up ssh keys for github: https://docs.github.com/en/authentication/connecting-to-github-with-ssh
- If you use VSCode, you won’t need to do this
- To use command line, you will
- Generate a public/private key pair (if not already)
- Upload public key to github
- Be careful, follow above instructions

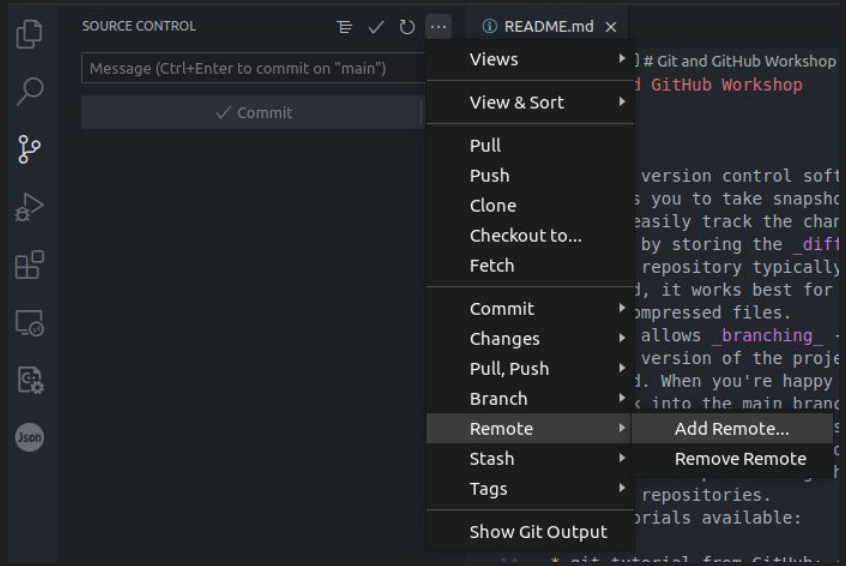
Make and commit some changes

- Use **git diff** to see your changes
- Stage and commit your changes
  - Git status
  - git add <...>
  - git commit -m <your commit message>
  - Or, commit using the source control tab in VSCode
  - See: https://www.gitkraken.com/learn/git/best-practices/git-commit-message



Remote

- Concept of a “remote”
  - Synchronise changes
- Named: by convention/default: “origin”
- If pulled from github: already set up



- *git remote -v*
  - List remotes with URL
- git remote add <name> <url>
  - Adds a new remote

- We can push/pull changes to/from remote
  - Keeps them syncd
- https://www.atlassian.com/git/tutorials/syncing

Push changes up to your github

- Push changes up to your github 'remote', and look at changes on github.com
  - git push origin main
- Check on github: see that the changes are there

Now, make a change on github

- git status
  - `it should say "up to date", because is doesn't know about the upstream changes`
- git fetch
- git status
  - Now, it should say there are changes to be pulled
- git pull
  - We should now have those changes locally

Branch

Make a new branch, check it out

* `git branch <branchname>`
* then: `git checkout <branchname>`
* Or: `git checkout -b <branchname>` (does both at same time)

Use `git branch` to list available branches

* Then make changes and commit them as before
* `git log` (or `git log --graph --oneline`)

Branch: push new branch

* push this branch up to github.com: `git push origin <branchname>`,
  or `git push --set-upstream origin <branchname>`

* Swap back/forth between branches:
    `git checkout main`
    `git checkout branchname`

Merge

Three ways this can go:

- “Fast forward”
  - No changes on destination branch

- Merge
  - Non-conflicting changes on both branch
  - Creates a new “merge commit”

- Conflict
  - Changes could not be auto-merged
  - Manually fix/decide, create new commit
  - Use a good text editor like VSCode

Conflict

If you are feeling brave: create a conflict

- change (+commit) same part of a document on both branches before merging

```
BR:git-workshop$ git merge newbranch
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
BR:git-workshop$
```

```
## git

Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
git is an awesom version control software.
=======
git is a great version control software.
>>>>>>> newbranch (Incoming Change)
It allows you to take snapshots (called commits) of your project, so
you can easily track the changes made to a project
```