October 2025
“git + Github”
And An AI Interview

git + github - zero to hero in 90 min
+ An AI interview movie

Agenda

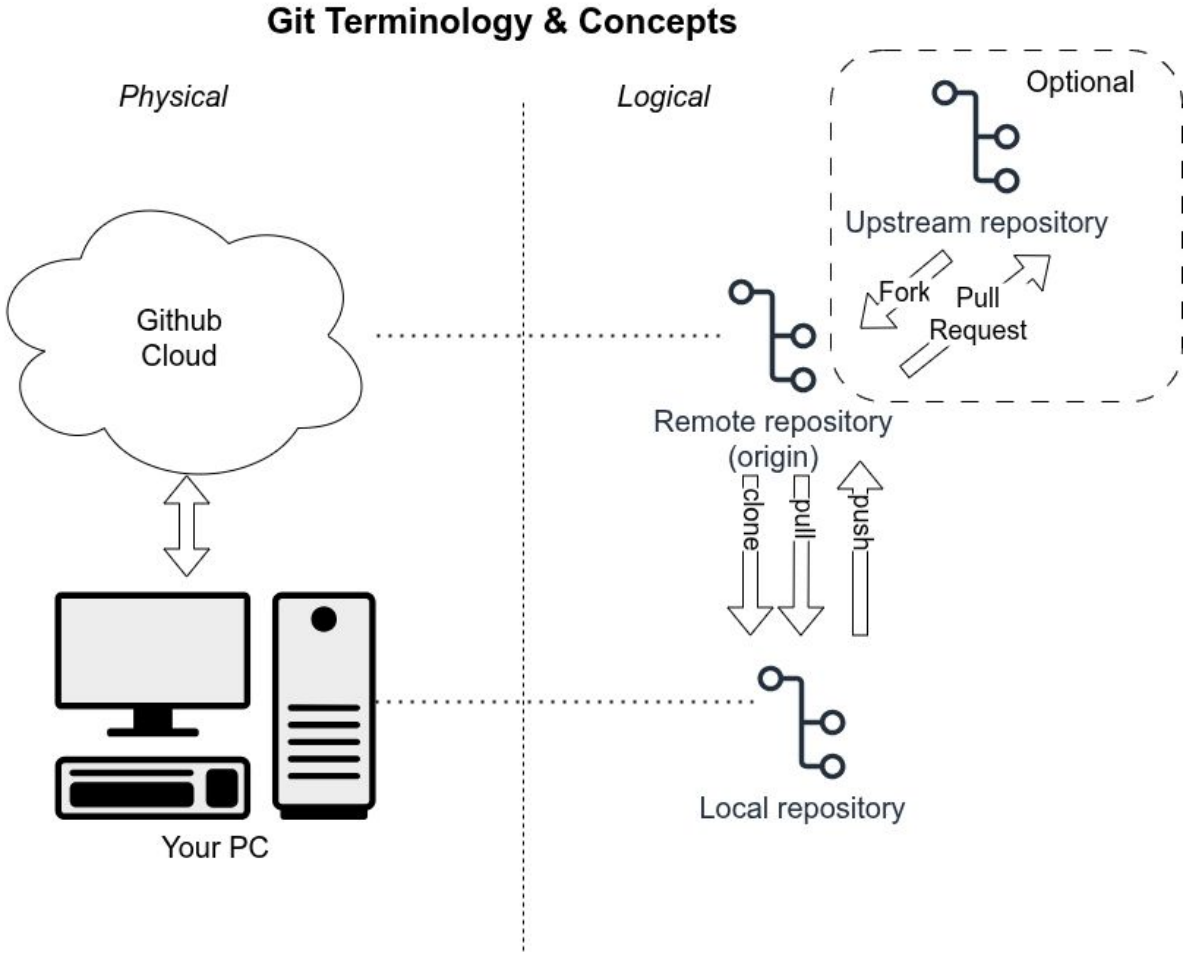
- DPRG News - RoboColumbus
- git + Github
- AI Interview Movie
- Lunch
- Show’n’tell

git + Github
Part 1

Basics + some advanced techniques

What is git + Github

- Github keeps a copy of your repository in the cloud
- git manipulates the local repository version history
    - Provides for syncing it with the cloud version (push/pull/clone commands)
- Github provides for creating new cloud repos, and interacting with optional upstream repos (fork, pull request)
- Git tracks diffs between versions
    - Diffs of text files keep commit sizes small
    - binary files supported (schematics, 3D models)



Why use git + Github? A ton of reasons...

Robotics centrally includes software/firmware. You need to use proper techniques in its development. (There's some overhead in managing the local, and optional remote repos.) Git enables good techniques, Here's how:

- Cloud backup of your code saves you if you lose your code folder on your PC.
  - It's a lot harder to delete a github repo than a directory.
- Save intermediate "working states" of your code, and easily get back to them for testing
- Switch between multiple configs - e.g. between two robot configs, or two sensor configs
- Avoid "multiple copies of code directories" syndrome.
  - Easily make changes to test something without changing the known-good version (difficult without multiple copies if not using git)
- Fearless refactoring or trying out new libraries/features.
  - Git encourages bold changes. Start a major refactoring on a branch and abandon it if it doesn't work out. Safety net gives you confidence to try big changes.

Why use git + Github (cont.)?

- Document changes (vs. documenting lines of code) with comments in the commit message.
  - Supports long-term project archeology - months or years later you can see not just what you changed but why.
- Sync your code directory between multiple machines. Get warnings about conflicting changes between them.
  - Scenarios: same codebase on PC and on Raspberry Pi used for different purposes
- Maintain a "Release" version that's always ready to run, while you develop on a branch.
  - Helps you quickly isolate whether a problem that arises in development is due to the SW change or if HW broke
- Maintain version control of HW files (3D cad models, PCB designs etc), as well as software & firmware.

Tools, Getting Started

Git is multi-platform - Windows, Linux, MacOS, x86 + Arm

- Install git on your platform.
  - You don't need the github CLI (gh). Use the website instead.
- If you use vs-code, install the git extension
- If you don't have a Github account, create one at github.com
  - You'll have to set up ssh keys and 2FA

Advanced Git Use

- git remote -v # display where the remote repo is
- .gitignore # configure files & directories that should be ignored by git
- git push -f # forced push makes remote match local repo
- git revert (before push & after push) # Back out a commit
- git cherry-pick <hash> # pull a single commit from one branch to another
- git blame <file> # Find out who last changed each line of a file
- git rebase -i HEAD~<n> # Squash n previous commits into a single commit by rewriting history
- git bisect # Seek to the commit that broke something

git + Github
Part 2: demos + branching

One more reason why you should use git for your software

When you use AI to generate code and write files, you really want a backup, that you can get back to if results are not as expected. Copying the entire directory everytime you want the AI to update files is not practical.

Get started on Github

- Create an account on github.com (you can sign up with Google, Apple, or email)
- Verify your email
- Generate a ssh key or Personal Access Token (PAT) and store it in your github account using these instructions:
  - https://www.geeksforgeeks.org/git/how-to-fix-support-for-password-authentication-was-removed/
  - Github does not accept password authentication - it has to be something more secure. PAT is ok. Ssh is more secure but (a little) more complicated
- Linux: alias gitk='gitk -a' —- Windows: consult Google

Configure git on PC

- Install git on your PC
  - Windows: git-scm.com/install
  - Linux: sudo apt install git
- Install meld on your PC
  - Windows: meldmerge.org
  - Linux: sudo apt install meld
- Run git commands from a shell:
- `git config --global user.email 'user email'`
  - This creates the .gitconfig file, which you can update in the next step

Getting Started: Edit .gitconfig in your home directory

The .gitconfig file in your home directory contains global settings and convenience aliases.

- Windows: \Users\<username>\.gitconfig —- Linux: /home/<username/.gitconfig

```
[user]
        email = paul.bouchier@gmail.com
        name = Paul Bouchier
[alias]
    s = status
    co = checkout
[diff]
    tool = meld
[difftool]
    prompt = false
[difftool "meld"]
    cmd = meld "$LOCAL" "$REMOTE"
[merge]
    tool = meld
[mergetool "meld"]
    cmd = meld "$LOCAL" "$MERGED" "$REMOTE" --output "$MERGED"
```

Basic Demos

Assumption: You have created an account

- Create & Clone new repo
- Copy some local content into new repo. Edit README.md
- View local repo status
  - git status + vs-code + gitk + git difftool
- Add to staged & commit and push it to github.

Advice:

- Commit when you reach a point that you consider “a significant accomplishment”, and to which you might want to return
- Push to Github whenever you commit

Create repo

Browse to github.com/<your_username>

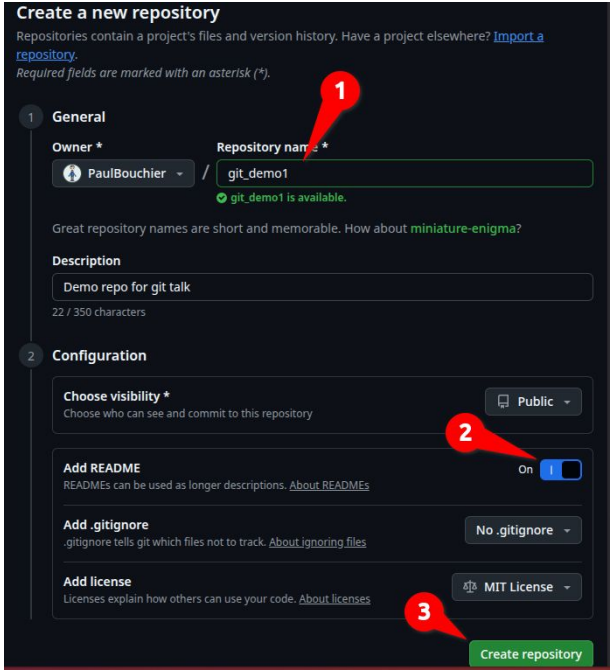
1. Click “Repositories” tab
2. Click “New”

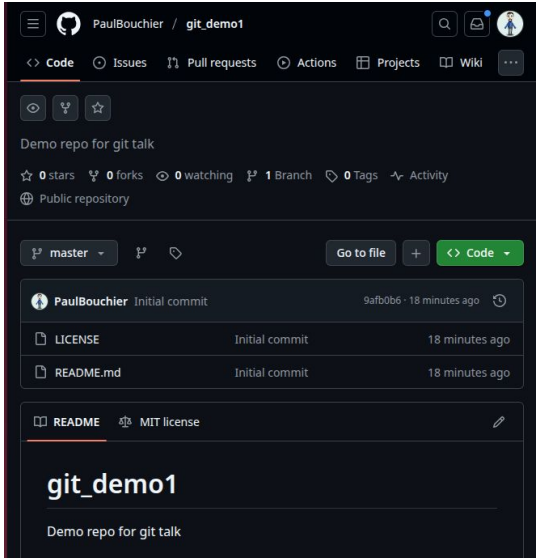
In the “Create New” page

1. Enter a name for your repo
2. Click “Add README” on
3. Click “Create repository

The page changes to show the contents of your new repo with some default files.

You have now created a repo in the cloud. Now you need to clone it onto your PC

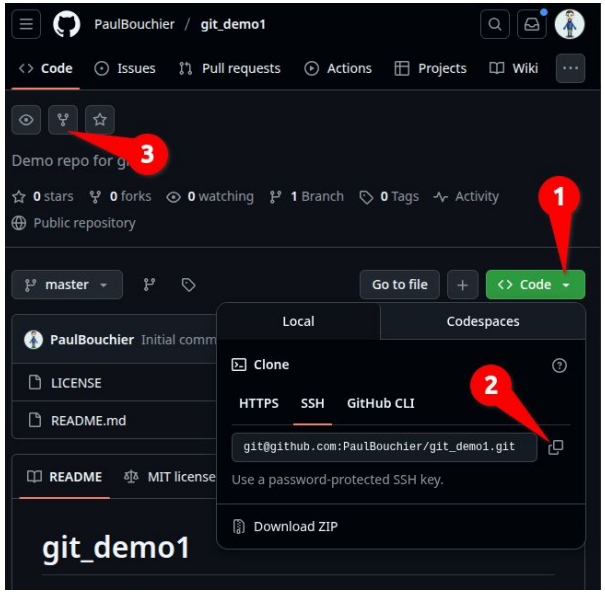




Clone Repo onto your PC

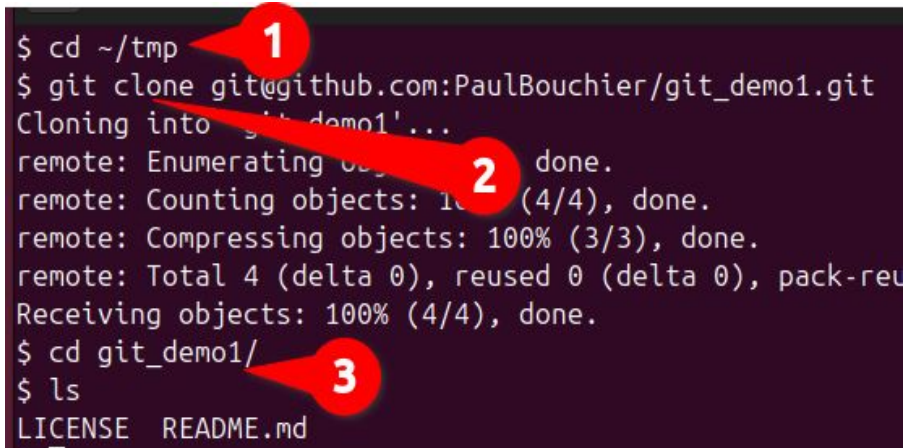
This procedure works for any repo on github, not just one you created

1. On the github repo page, click the arrow next to the “Code” button
2. Click the “copy” icon next to the URL to copy the URL you will clone onto your PC
3. Fork any repo with this button



Open a terminal window

1. cd to the folder you want to clone into (e.g. src)
2. Type “git clone <paste URL>
3. Show contents of <repo_name> folder



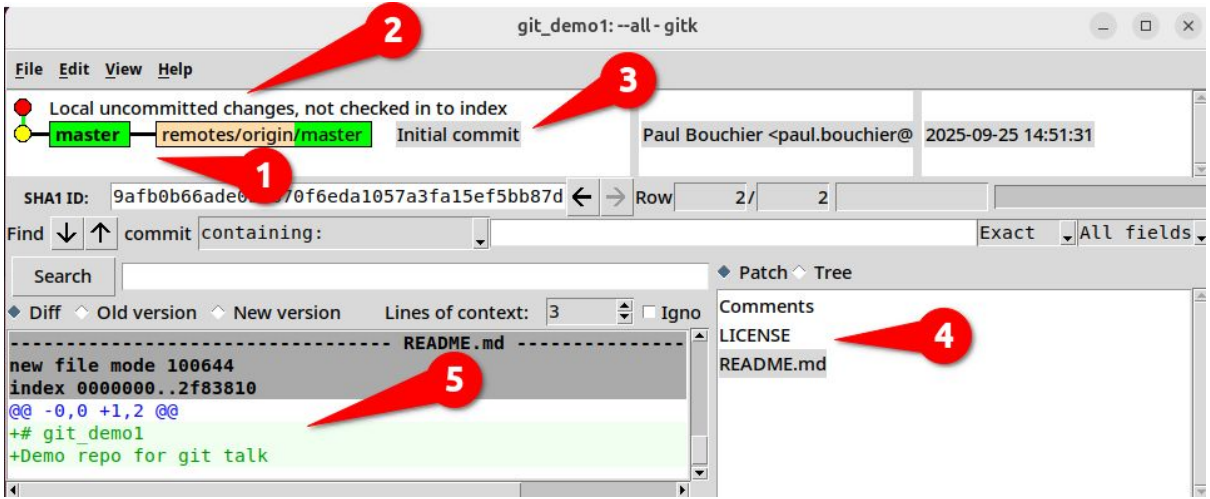
You have now cloned the remote repo into a local copy having the same folder name as your repo

Making and inspecting changes - gitk

Copy the files you want under into new repo. Edit README.md

Launch gitk -a

ALWAYS USE -a flag



Observe

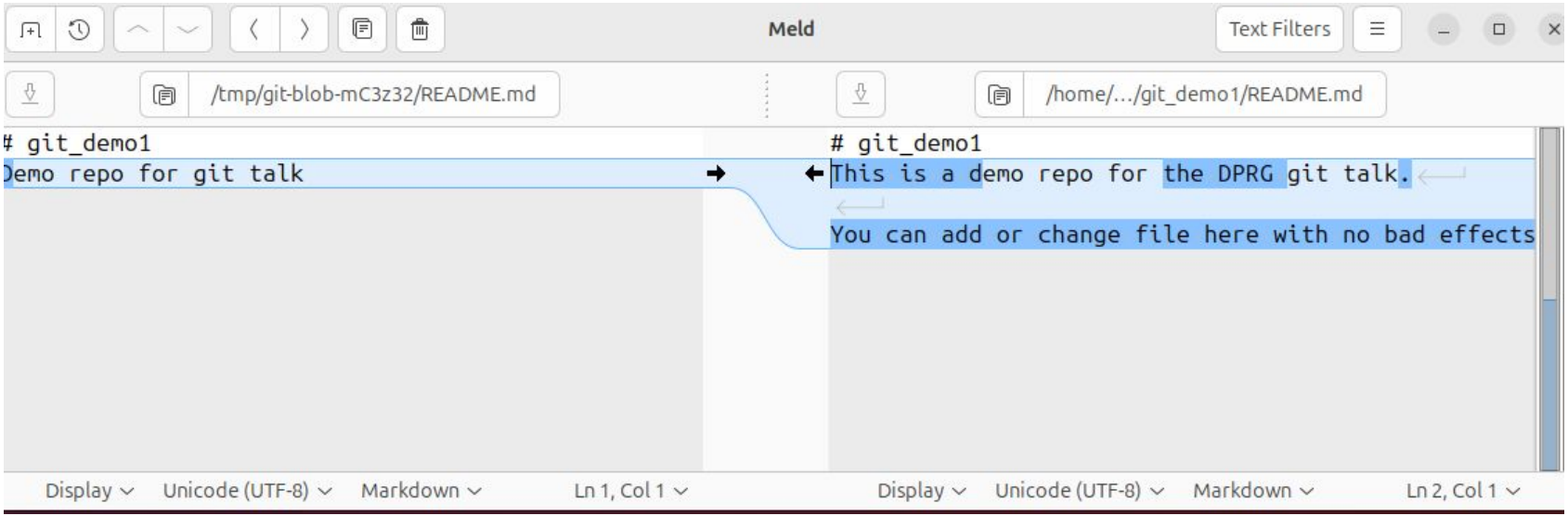
1. Local and remote master are the only commit, and are the same
2. There are uncommitted changes
3. Shaded commit comment shows selected commit. Hash in SHA1 ID
4. Lower right pane shows files changed in selected commit
5. Lower left pane shows changes to each file that was changed

Resize gitk panes, window. Click on other lines in git history. Run gitk in other repos you have downloaded.

Inspecting changes - git difftool

In command window, type “git difftool”

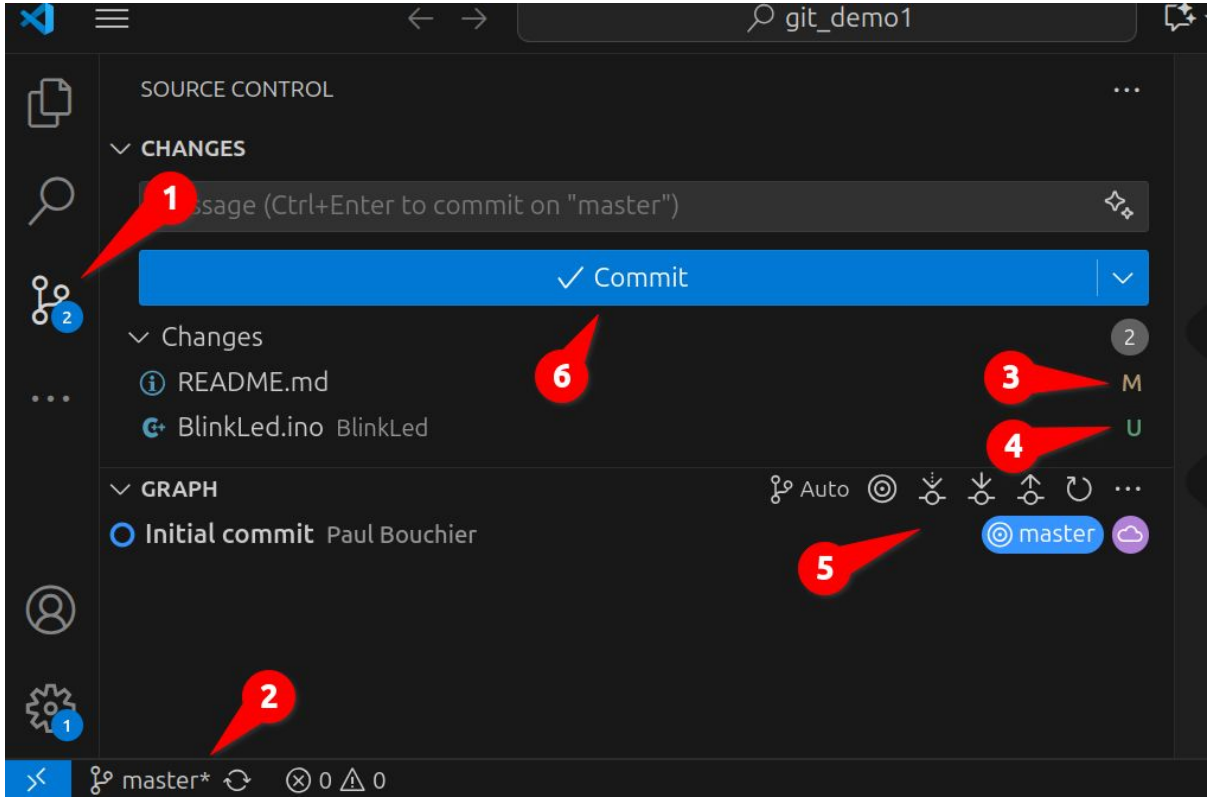
Observe meld shows changes to each modified file



Inspecting changes - vs-code

Launch vs-code in the repo.

1. Select the git extension
2. You are on the master branch and it has been modified
3. One file change has been modified
4. The other file change is uncommitted
5. Actions are available to pull/push
6. You can commit these changes

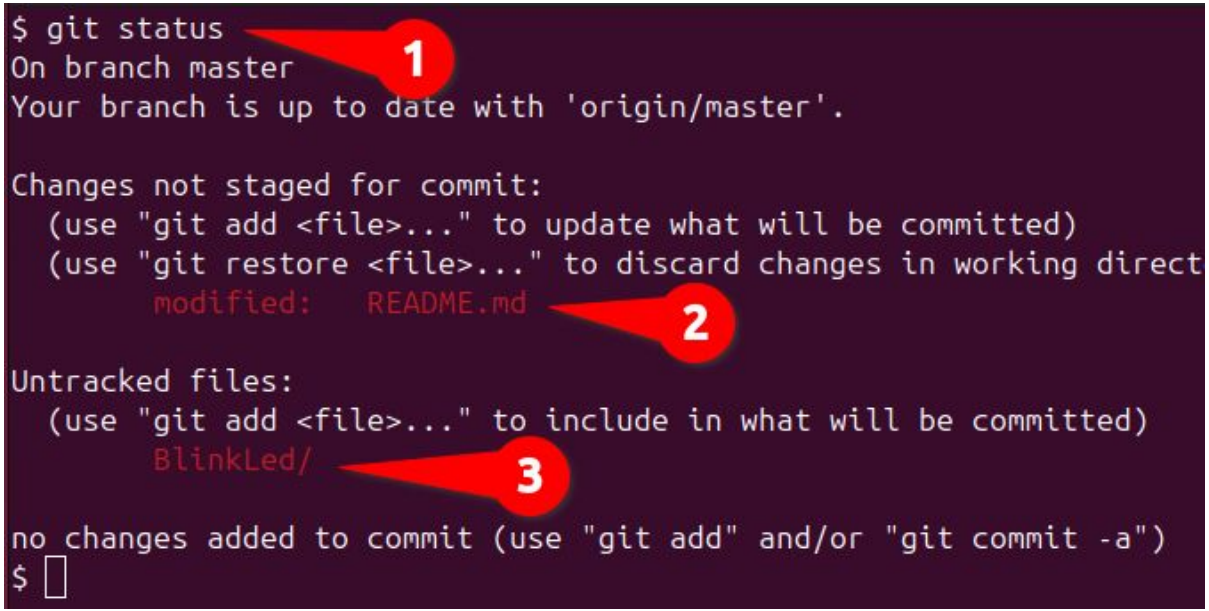


Disclaimer: I don't use vs-code for git, and have no experience with it. I prefer command line + gitk

Inspecting changes - CLI

In the command window:

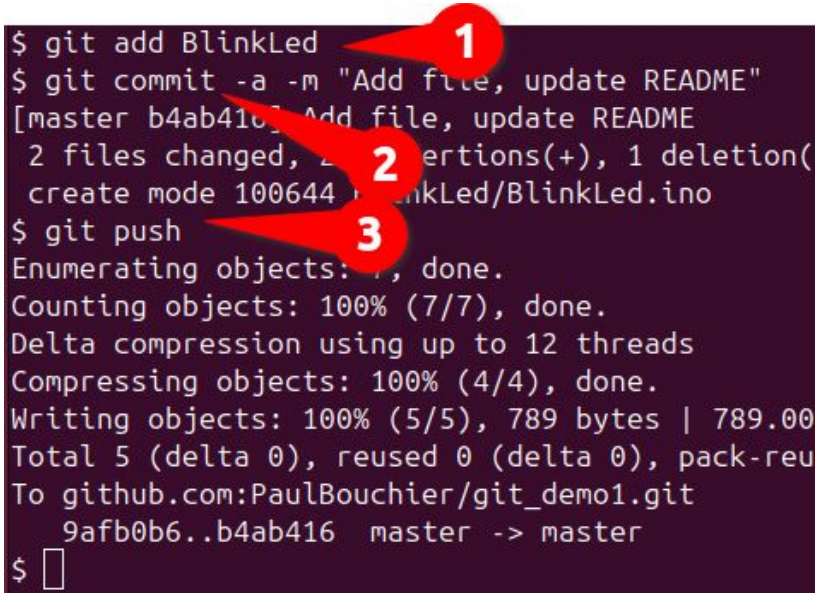
1. Type “git status”
2. Observe 1 file changed, not staged for commit, with instructions for unstaging
3. Observe an untracked folder (which contains our new file) with instructions for adding

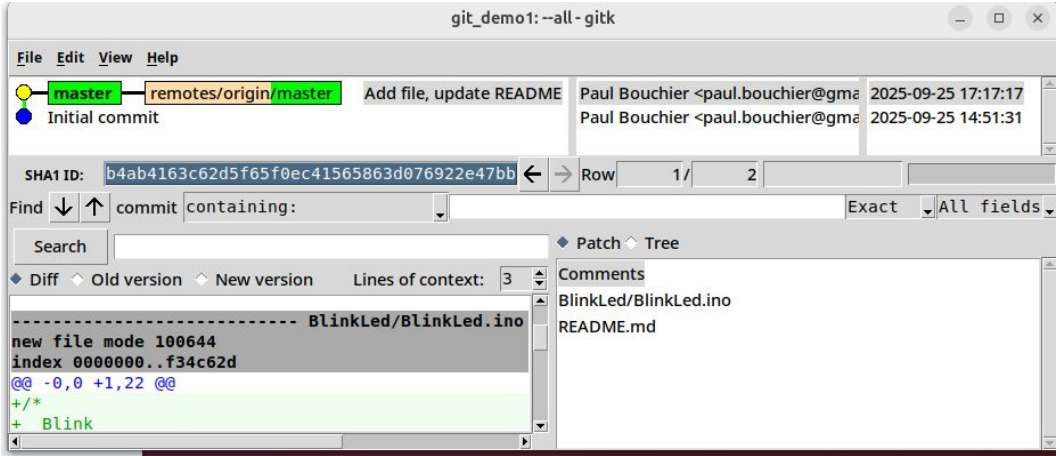


Stage, commit & push

In the CLI, type:

1. git add <new files/folders>. Update gitk and check: File ↠ Reload
2. git commit -a -m “<comment>”. Update gitk and check
3. Git push. Update gitk and check





What's a commit?

1. Conceptually: an alternate reality in the filesystem. Files and folders may wink in/out or change between commits
2. A recorded set of changes to files, including file creation, deletion, which can be restored with a checkout

- Each commit has a hash that uniquely identifies the file changes from its preceding commit
- Commits are sequential, each one modifying the previous one
- Making a commit:
  - Edit files. Create files/folders. git rm files to be deleted
  - Stage the changes (choose which changes to record in git)
  - Commit the changes with a comment as to what & why
  - Push the commit to github, to preserve the changes
- Check out a commit (file system restored to that point) using its hash
- DEMO with gitk

Branches - why?

Build off an existing checkpoint in a way that

- Is easy to back away from
- Doesn't commit you to the changes you're making
- Enables you to easily go back to your checkpoint

Example 1: you just completed your motor PWM code and it works. Now you want to add wheel encoder interrupts, but you aren't 100% sure you understand all the nuances. If you run into problems with the interrupts, you'd ideally be able to revert to the working PWM code and make sure the HW is still working.

Example 2: You want an AI to build your wheel encoder code. Will it work?

Solution: Create a branch and develop on it. You can abandon it if you don't like it. The branch name refers to the most recent commit (called HEAD)

Branches - what?

1. A chain of commits (alternate realities in your folder tree) that's maintained by git
2. A name to refer to the chain of commits, and treat the chain as a unit (e.g. to rename, merge etc)

- When you check out a branch, files & filenames may change, per the commits on the branch. File content may change. The entire directory tree undergoes changes to make it match
- Branches can be added, deleted, renamed, merged to another branch, rebased on another branch
- There are 2 versions of a branch: local and remote. Try & keep them synced

Branches - how?

Use gitk frequently when working with branches!

- git checkout <existing_branch_name> # switch to named branch
- git checkout -b <new_branch_name> # create a new branch, which branches at the currently-checked out commit
- git branch -a # display all branches, * alongside what's checked out

Merging

Fast-forward merge: HEAD of current branch is advanced to the HEAD of merged branch. Requires a linear advance.

- git merge <existing_branch_name> # merge the branch into the currently checked-out branch
- git rebase <other_branch_name> # Change the commits of the current branch so that it becomes modifications to other_branch_name

I recommend rebase then fast-forward merge over plain merge - history is simpler to follow

Conflicting pushes & 3-way merge & other issues

- git mergetool demo
- .gitignore
- Squash demo: checkout a branch and squash it

References

Repo with this talk's slides: https://github.com/dprg/2025_git_talk

https://github.com/dprg/2025_git_talk

Thomas Messerschmidt

An interview with an AI (gone wrong)

Show’n’tell