Version Control

What is version control and what’s it good for?

It’s a system for keeping track of changes to a set of files over time.

This allows us to do things like:

- Keep a record of all versions of a codebase
- Run a specific version of code - e.g. on a server
- Pick any version as a new starting point (branching) - try different experiment versions
- Collaborate with others more easily

Why is this good for research?

Beyond collaboration, the key benefit to researchers is code reproducibility.

There is a simple formula for making experiments reproducible:

`same code + same inputs = same output` *

Version control solves the ‘same code’ part of this equation. If you give someone a git repository, and tell them which commit you used to produce a result, then they will be able to checkout that commit and run it again.

* There are a couple of nuances which will we’ll cover later.

Intro to git

We recommend learning to use git through the command line at first. GUIs are great but tend to obscure what’s going on.

The main operations we’ll cover

- init - create a repo
- add - staging area
- commit - history
- push/pull - remote
- branch/checkout/delete - branching

This is a great resource: https://git-scm.com/book/en/v2

Some key concepts in git

There are lots of quirks to git, but the core ideas are quite simple.

- **Repository** - your project. Think of it as a database with every commit ever made stored in it.
- **Commits** - a savepoint for the state of your repository. It’s literally just a copy of all of your files.
- **Staging area** - where git looks to find changes for the next commit.
- **Branches** - different lines of development work. Work doesn’t always happen in a straight line. Sometimes you go back 2 weeks and from there go in a different direction.

git init

Usage:

```
git init
```

This command initialises a new git repository. You won’t have to use it very often - once per project.

To use it you simply move into the directory that you want to make a git repo, and run the command.

git add

Usage:

```
git add .
```

This adds files to the staging area.

The staging area is used to tell git which changes you want to include in the next commit.

The ‘.’ simply means ‘every file in this directory’. You can also pass the path to a folder or file and only the changes in that path will be staged.

git commit

Usage:

```
git commit -m “a commit message”
```

This command creates a commit from everything in the staging area.

Once you have created a commit, it acts as a ‘save point’. You will be able to go back to this state at any time and branch off of it.

You should try to create a descriptive commit message that describes what the commit actually does. For example:

- Add new inverse transformation function
- Fix issue 247

Keep commits small and make them regularly!

git push/pull

Usage:

```
git push

git pull
```

Push will update the remote with your local commits.

Pull will update your local repo with commits from the remote.

This is simplest if your local copy is ‘clean’ before you run the commands. I.e. there are no uncommitted changes.

git checkout

Create a branch:

```
git checkout -b new_branch_name
```

Checkout an existing branch:

```
git checkout branch_name
```

You use these commands to create and move between branches. You should be using these a fair bit!!

Best practice is to always work on a branch, and never against the mainline branch. When you want to add a feature or bug, you should create a new branch for that change. When you are happy with it, you should merge that branch back into the mainline.

This ensures that the mainline always ‘works’ and does not end up in a half broken state.

Mental model for collaborating with git

- Git is a distributed VCS system. This means that there are lots of copies of the repository and they're all the same as each other.
- Git allows you to copy commits between different copies of the same repository.
- This means that we can use a central copy of the repository to share changes with other people.
- This is what GitHub is for!
- When you create a repository on GitHub, it is the same as the one you have on your laptop.

Intro to github

GitHub is just a service for hosting git repositories.

The primary reason we host a git repository is so that we can collaborate with other people.

We all treat the ***'remote'*** copy of the repo (the one on GitHub) as the ***'origin'***. I.e. the source of truth.

The best way to learn more about it is to read their documentation:

https://docs.github.com/en/get-started

Creating a repo

We will walk through this in the practical:

The best reference is their documentation:

https://docs.github.com/en/get-started/quickstart/create-a-repo

GitHub Workflow

This workflow is most relevant when you are working with others. It is still a good discipline to follow it when it's just you though. It will help keep things tidy and make it easy to get anyone else involved.

- Update local from remote
- Create new branch
- Make changes
- Commit and push
- Open pull request
- Review changes - ideally get a second opinion!!
- Merge and then delete branch

The best reference is their documentation:

https://docs.github.com/en/get-started/quickstart/github-flow

Local git workflow

```
git checkout -b new_branch # New branch

git add .  # Stage some changes

git commit -m “describe your changes here”

## If we want to merge back to main now

git checkout main

git merge new_branch


## Delete a local branch

git branch -D new_branch
```

GitHub workflow command cheat sheet

```
git pull

git checkout -b new_branch

git add .

git commit -m “describe your changes here”

git push -u origin new_branch
```

You would then log on to GitHub and create a pull request in there.

Working with remotes

```
git remote -v (list the remotes)

git remote add origin url

git remote set-url origin url
```

Code Review

- Code review is getting someone else to take a look at your code.
- It is one of the most effective practices there is to improve code quality and to learn from each other.
- If you can establish a culture of code review in your cohort, I guarantee you will end up producing code that is significantly better than your peers who do not.

How to Review Code

- Do not make it personal, writing good code is extremely difficult.
- The idea is to learn from each other, and to help each other produce better work than you could alone.
- It is about asking questions, not looking clever.
- If it is not clear how something works, just ask why
    - *It looks like this function is meant to do x, but it's not clear to me how it accomplishes that. Can you explain it please?*
- If someone else can't understand your code, you will not be able to in 6 months either. You should modify it so that it is easy for them to read.

Best practices

- Commit often
- Keep main clean - always develop on a branch
- Merge branches back into the mainline often
- Get someone else to review your pull requests