Git/GitHub cheatsheet

Things to think when publishing code

- **Public or Private?** - What level of visibility should your project to have? Open code promotes reproducibility and impact but be sure to consider and consult any other contributors
- **README.md** - The ‘front page’ of a repository. GitHub will automatically render the markdown in this file nicely
- **Licence** - Tell other people what they can and cannot do with your code. The College recommended licence is BSD 2- or 3-clause
- **INSTALL.md** - If your project is complex to setup and install be sure to include detailed instructions
- **CITATION.cff** - Show how your project should be cited by others
- **CONTRIBUTING.md** - Tell potential contributors the kind of work you’re interested in and the process they should follow

Glossary

- **Repository/Repo** – A project that is being managed by Git
- **Commit** – A set of changes recorded in the history of a project
- **Staging Area** – The location where file changes are recorded to prepare for inclusion in a commit
- **Working Tree** – The visible copy of files in a project that you view and edit as usual
- **Branch** – A label used for a set of commits with a particular purpose
- **Merge** – Combine the changes from another branch (or commit etc.) into the current one
- **Merge Conflict** – A state that occurs when merging automatically fails because two sets of changes are incompatible. In this case, the changes must be resolved manually.
- **main** – The default name given to the starting branch of all repositories
- **HEAD** – The current commit that the working tree is based on
- **Local** – A repository copy stored on the computer where you are working
- **Remote** – A repository copy stored elsewhere, usually a hosting service like GitHub
- **origin** – The default name given to a repository’s remote
- **Upstream** – The remote repository with which a local repository is associated
- **Tracking** – Used to describe a branch in a local repository which is matched with a branch in a remote repository
- **Fork** – A copy of a repository on GitHub that is owned by a different GitHub user
- **Origin** – The default name used by Git for a configured remote repository
- **Pull Request (PR)** – A GitHub feature which requests that changes from a fork be incorporated into the original repository
- **Continuous Integration (CI)** – A software development practice which involves running automated checks to ensure code contributions meet certain criteria. An example of a CI system is GitHub Actions.
- **Semantic Versioning** – A versioning scheme where the different numbers in a software version (e.g. v1.2.3) have a particular meaning

(Continued overleaf.)

Git Command Cheat Sheet

- **git config** – Change (or view) the settings that Git uses
- **git init** – Create a new repository in the current directory
- **git status** – High-level overview of changes made since the last commit
- **git stage** – Stage a file (or changes made to a file) for inclusion in the next commit
- **git add** – See "git stage"
- **git commit -m "COMMIT MESSAGE"** – Create a new commit including all staged file changes
- **git commit --amend** – Incorporate further changes into the last commit and/or edit the commit message
- **git log** – Display the commit history of a repository
- **git diff** – Show in detail the changes made in the working directory since the last commit
- **git reset --soft HEADˆ** – Remove the last commit from the history of a repository
- **git revert --no-edit COMMIT_HASH** – Create a new commit that undoes the changes of the specified commit
- **git branch** – Report on the existing branches in a repository
- **git branch BRANCH_NAME** – Create a new branch
- **git branch -D BRANCH_NAME** – Delete a branch (be careful with this one!)
- **git switch BRANCH_NAME** – Update the position of HEAD to a new branch
- **git switch --detach COMMIT_NAME** – Update the position of HEAD to a new commit
- **git checkout** – See "git switch"
- **git merge --no-edit BRANCH_NAME** – Merge BRANCH_NAME into the current branch
- **git rebase NEW_BASE** – Rebase current branch onto NEW_BASE
- **git remote add origin REPOSITORY_URL** – Configure a local repository with a remote with the label 'origin'
- **git push** – Synchronise changes in the current local branch to its upstream branch
- **git push --tags** – As above, but also push any tags you've created to the remote
- **git pull** – Synchronise changes in the upstream branch to the current local one
- **git clone REPOSITORY_URL** – Create a new local repository that is a copy of remote one
- **git tag TAG_NAME [COMMIT_HASH]** – Create a new tag at the specified commit (or at HEAD, if not specified)

For a more exhaustive description of the various Git commands and their options, you can consult Git's online documentation.

Further Help

This course was developed by the Research Computing Service (RCS) at Imperial College London, in particular by the Research Software Engineering (RSE) team.
The RSE team are a part of Imperial ICT combining specialist knowledge in software engineering with extensive experience in research. The team works with academic groups on a wide range of projects whilst also organising community events and training (such as this course) for the benefit of the research community. You can find out more at the RSE team website and the Imperial Research Software Community website. You can also consult the expertise of the RSE team by booking a code surgery appointment.
You may also be interested in attending another course developed by the RSE team titled **"Essential Software Engineering for Researchers"** which can be booked via the Graduate School.