Lecture 5 — Version control with Git

C1 Research Computing — MPhil in Data Intensive Science, Michaelmas 2026

Boris Bolliet

University of Cambridge

Outline

- Commits and the staging area
- Branches, merges and conflict resolution
- Remote repositories and collaborative review
- Ignored files, secrets and large data
- Inspecting history, tagging and reversing changes
- Live demonstration: local history and a merge conflict

What is Git?

- A repository is a project with its Git history; a clone is a local copy of the repository, including that history.
- Git is distributed: each clone contains the history and can record commits independently.
- Linus Torvalds began developing Git in 2005 for Linux kernel work, after free use of BitKeeper ended.
- Git is the version-control system. GitHub, GitLab and Bitbucket host remote Git repositories and provide code-review and collaboration tools.

Version control for research code

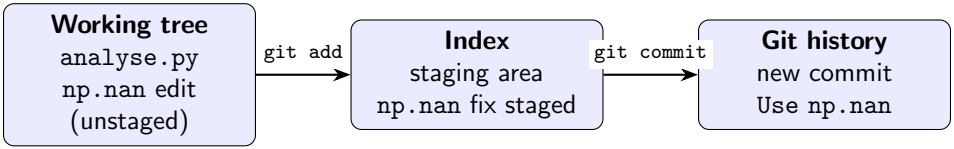
- Git records changes to selected project files as commits.
- A commit identifies a project state that can be inspected or restored.
- Git does not record an interpreter, installed packages or external data unless those are represented in project files.
- Example project files: `analyse.py`, `requirements.txt`, `.gitignore`.

Commit snapshots

- A commit is a saved snapshot of selected project files, with a message, author, date and parent commit or commits.
- A parent links to an earlier state: an initial commit has none, an ordinary commit has one, and a merge commit may have two.
- Each commit has an identifier. A short prefix is commonly shown in logs.
- The repository stores commits and the file contents they reference in `.git/`.

Commits and snapshots

Attractor: staging a compatibility fix



- NumPy 2 removed `np.NaN`; the fix in `analyse.py` uses `np.nan`.
- Working tree: editable project files on disk. Index (staging area): file contents selected for the next commit.
- `git add` copies the chosen file version to the index; `git commit` records that snapshot.
- After another edit, `git restore analyse.py` restores the indexed version.

Recording changes | git restore

✎ Updating attractor code

```
git clone https://github.com/borisbolliet/attractor.git attractor-git-demo
cd attractor-git-demo
git config user.name "Ada Lovelace"
git config user.email ada@example.org
# In analyse.py, replace np.NaN with np.nan
git diff -- analyse.py
git add analyse.py
git diff --staged -- analyse.py
git commit -m "Use np.nan"
git status --short
```

- `git clone` creates a local repository copy with project files and history.
- `git config` sets the name and email recorded on commits made in this repository.
- `git diff` compares the working tree with the index; `git diff --staged` shows the next commit's contents.

attractor | git add and git commit

Inspecting staged and unstaged changes

```
# In analyse.py, change MIN_GAP from 3.0 to 3.5
git diff -- analyse.py      # working tree compared with index
git add analyse.py          # stage the latest file content
# Change MIN_GAP from 3.5 to 4.0
git diff -- analyse.py
git diff --staged -- analyse.py
git status
git status --short
# MM analyse.py: staged and unstaged edits
git add analyse.py
git commit -m "Test peak spacing"
```

- `git status` reports the current branch, staged changes, unstaged edits (not yet in the index) and untracked files (not yet added to Git).
- In `git status --short`, the two columns show index and working tree; `MM` means both contain changes.

git diff and the index

Commit history

```
git log
git log --oneline --graph --decorate
```

- `git log` lists commits, newest first, with their identifiers, authors, dates and messages.
- A branch is a name pointing to a commit; `HEAD` marks the current branch.
- `--oneline` shortens each entry; `--graph` draws parent links; `--decorate` labels branches and `HEAD`.

git log

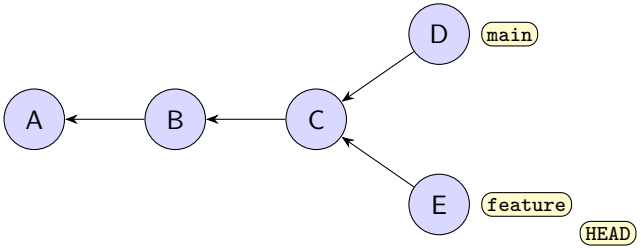
Tags and analysis provenance

```
git tag -a analysis-v1 -m "Code used for Figure 2"
git show --stat analysis-v1
```

- An annotated tag names a commit and records a message, creator and date.
- A tag can identify the code version used for a reported figure.
- The data and environment also need records in the repository or a linked archive.

git tag

Branches and `HEAD`



- A branch is a name that points to a commit and advances when a commit is made on it.
- `HEAD` identifies the currently checked-out branch in this example.
- Commits A–C are shared history; D and E are separate lines of development.

Branches

Creating and switching branches

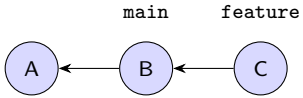
```
git branch                  # list local branches
git switch -c feature       # create and switch to feature
git switch main             # switch to main
git branch -d feature       # delete a merged branch
```

- `git switch -c` creates a branch at the current commit.
- Switching branches updates the files to the selected branch's committed state.
- Git stops if local changes would be overwritten by the switch.

git switch | git branch

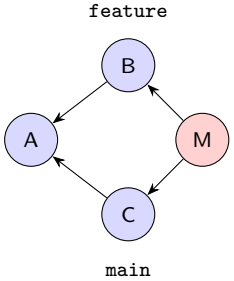
Merging branches

Fast-forward



Move `main` to C; Git creates no commit.

Merge commit



M has two parents: B and C.

```
git switch main     git merge feature
```

Merging

Merge conflicts

A conflict requires a choice when Git cannot combine the changes automatically.

```
<<<<<<< HEAD
step_size = 0.005
=======
step_size = 0.02
>>>>>>> larger-step
```

- `HEAD` marks the current branch; the named branch follows `=======`.
- The conflict may involve overlapping edits or file changes Git cannot reconcile.
- Edit the file, remove the markers, stage the resolution and commit the merge.

git merge

✎ Resolving a parameter conflict

```
git switch -c larger-step
printf 'step_size = 0.02\n' > settings.py
git add settings.py && git commit -m "Test larger step"
git switch main
printf 'step_size = 0.005\n' > settings.py
git add settings.py && git commit -m "Test smaller step"
git merge larger-step
# Edit settings.py and remove the conflict markers.
git add settings.py
git commit -m "Resolve step-size conflict"
```

- The example changes the same setting on two branches.
- The resolution is a scientific choice; Git does not select a step size.

Merge and resolve conflicts

Remote repositories

Remote names are saved URLs. `git remote -v` lists them; `origin` is usually added by `git clone`. Assume `main` is checked out in both examples.

Fetch, inspect, then merge

```
git fetch origin
git log --oneline main..origin/main
git merge origin/main
```

`git fetch` downloads commits and updates `origin/main`. Local `main` and project files stay unchanged until `git merge`.

Pull and integrate

```
git pull origin main
```

Fetches remote `main` and integrates it into the checked-out branch. Git uses merge by default; configuration can select rebase.

`git push -u origin feature` publishes `feature` and records its upstream branch for later pushes.

git fetch | git pull | git push | GitHub authentication

Forks, clones and pull requests

- A **fork** is a hosted copy of a repository under another account.
- `git clone URL` copies a repository and its history to the local machine; the remote is conventionally named `origin`.
- A **pull request** asks maintainers to review and merge a proposed branch; GitLab calls this a merge request.

$$\text{source repository} \longrightarrow \text{fork} \longrightarrow \text{clone} \longrightarrow \text{branch} \longrightarrow \text{pull request}$$

- Review takes place on the hosting service; new commits pushed to the branch update the request.

GitHub pull requests | GitLab merge requests

Ignore rules

```
# .gitignore
.venv/
__pycache__/
*.py[cod]
.env
results/
```

- A tracked file is recorded by Git; an untracked file has not yet been added. Matching untracked files are omitted from `git status` and `git add .`.
- Ignore rules do not stop Git tracking a file already in the repository.
- Commit `.gitignore` so the rules apply to every clone of the project.

gitignore | Ignoring files on GitHub

Secrets and research data

- Keep API keys and passwords out of commits; store local values in an ignored file such as `.env`.
- Deleting a secret in a later commit does not remove it from earlier history.
- If a credential is pushed, revoke or rotate it immediately.
- Commit small, redistributable data when useful; GitHub blocks ordinary Git files larger than 100 MiB.
- Git Large File Storage (LFS) keeps large file contents outside ordinary Git objects; commits contain small pointer files.

Secret scanning | GitHub file-size limits

Unstaging and discarding edits

```
git restore --staged analyse.py   # remove from index; keep working-tree edits
git restore analyse.py            # replace edits from the index
```

- The first command changes the index only.
- The second command discards unstaged edits to the file.
- Inspect with `git status` and `git diff` before restoring.

git restore

Reverting a committed change

```
git log --oneline
# 3f2a1c9 Fix step-size conversion
git revert 3f2a1c9
git log --oneline
```

- `git revert` creates a new commit that reverses the selected commit's changes.
- Existing history remains in place, including on a branch already shared with collaborators.
- A revert can require conflict resolution if later changes overlap.

git revert

Summary

- `git status`, `git diff`, `git add` and `git commit` show and record selected changes.
- Branches provide separate lines of development; merges combine them and may require a manual resolution.
- Tags identify commits; remotes exchange them, and pull requests support review on GitHub or GitLab.
- Ignore local environments and secrets; use `restore` for local edits and `revert` for a committed change.