ChatGPT

Advanced Git Productivity Guide for PhD Researchers

Monolithic-Repository Edition – Extended
Version 1.1 – June 8 2025

---

0 · Purpose & Audience

This guide is written for doctoral candidates, post-docs, and research engineers who operate in highly interdisciplinary projects yet store *everything*—source code, LaTeX manuscripts, experiment configs, data preprocessing scripts, figures, and even small datasets—in **one large, single-repository codebase**.

> **Why a monorepo?**
> • Atomic cross-component changes (API tweak ↔ paper diagrams).
> • Simpler CI/CD—one pipeline, one set of hooks.
> • Unified provenance chain for reproducibility audits and scholarly peer review.

Our goal is to convert raw Git tooling into a **repeatable research factory** that guarantees: 1. **Velocity** – rapid ideation and branching experiments.
2. **Integrity** – cryptographically signed, policy-compliant commits.
3. **Provenance** – ability to replay *any* published figure or table via commit hash + config.

---

1 · Baseline Configuration

A robust `~/.gitconfig` cuts cognitive overhead. Start with minimal defaults, then layer project-specific settings in `.git/config`.

1.1 Identity & Signing

```
[user]
    name        = <Firstname Lastname>
    email       = <netid@university.edu>
    signingkey  = <GPG-KEY-ID>
[commit]
    gpgsign     = true          # enforce signature
[gpg]
    program     = gpg
```

Tip ▶ University mail relays sometimes strip signatures—verify via `git log --show-signature`.

1.2 Core Behaviour

```
[core]
    editor          = nvim      # any modal editor works
    autocrlf        = input     # preserve LF on *nix, convert CRLF on
Windows input only
    whitespace      = cr-at-eol,trailing-space,space-before-tab
    fsmonitor       = true      # speed up status on giant repos (requires
git v2.46+)
[push]
    default         = current   # reduces accidental main pushes
[pull]
    rebase          = false     # prefer merge unless explicitly rebase
[merge]
    conflictstyle   = diff3
[alias]
    lg  = log --graph --oneline --decorate --all --color
    ls  = log --stat --abbrev-commit
    fp  = fetch --prune
    rc  = rebase --continue
    amend = commit --amend --no-edit
    wip   = "!git add -A && git commit -m 'wip: snapshot'"
```

1.3 Large Files & Data Assets

```
[filter "lfs"]
    clean  = git-lfs clean -- %f
    smudge = git-lfs smudge -- %f
    required = true
```

Track datasets > 20 MB:

```
git lfs track "data/**/*.npz"
```

---

2 · Hooks – Automated Quality Gates

Hooks transform your laptop into a *pre-review CI node*.

Hook	Phase	Typical Payload	Time Budget	Failure Policy
`pre-commit`	Index → Object	Code style (Black + Ruff), MyPy strict mode, unit tests, Jupyter `nbstripout`, `nbQA flake8`	≤ 7 s	Block commit

Hook	Phase	Typical Payload	Time Budget	Failure Policy
`commit-msg`	Msg buffer → Object	Regex `^(feat|fix|docs|test|perf|refactor)([^]*\))?: .+`	< 1 s	Block commit
`prepare-commit-msg`	Optional edit	Insert branch ticket ID, pair-programmer initials	negligible	Auto-modify
`pre-push`	Local → Remote	`pytest -q`, `bandit -ll`, secrets-scan, doc build	≤ 30 s	Block push
`post-merge`	Remote → Local	`pip install -r requirements.txt`, `dvc pull`, regenerate ctags	async	Warn only
`pre-rebase`	Safety gate	Check un-pushed commits, stash untracked, run `git status --porcelain`	2 s	Block rebase

2.1 Installing with `pre-commit`

```
pip install --user pre-commit
pre-commit install
pre-commit autoupdate        # quarterly bump
```

Pin revisions for deterministic builds:

```
repos:
  - repo: https://github.com/PyCQA/isort
    rev: 5.13.0
    hooks: [{ id: isort, args: [--profile, black] }]
```

2.2 Example Dynamic Metadata

```
# .git/hooks/prepare-commit-msg
branch=$(git symbolic-ref --short HEAD)
case $branch in
  feat/*|fix/*|exp/*)
    sed -i "1s/^/[$branch] /" "$1";;
esac
```

---

3 · Diffing – High-Fidelity Reviews

Efficient diffing surfaces semantic, not accidental, changes.

Scenario	Command	Rationale
Audit word-level edits	`git diff --word-diff=color -U5`	highlights variable names / prose tweaks
Structural move detection	`git diff -M90% -C`	detect renames, copies; avoid noise in refactor PRs
Large refactor readability	`git diff --patience`	patience algorithm groups coherent blocks
Suppress whitespace noise	`git diff -w --ignore-blank-lines`	skip re-indents
GUI vs TUI	`git difftool -t meld` or `nvim -d`	side-by-side review

Tip ▶ To view **in-progress** staged vs working: `git diff --cached`.

---

4 · History Manipulation – Controlled Rewrites

History is your dataset of time-stamped decisions.

4.1 Interactive Rebase (local only)

```
git rebase -i --autosquash HEAD~11
```

- `fixup!` commits auto-collapse against their targets.
- Use `exec pytest -q` lines to run tests mid-rebase.

4.2 Deep Surgery with `git-filter-repo`

Use when legal or privacy policy demands:

```
git filter-repo --path data/raw/patient.csv --invert-paths \
                --commit-callback 'commit.message += b"\n[redacted-pii]"'
```

Post-surgery:

```
git repack -Ad && git prune --expire now
```

4.3 Reflog-Driven Recovery

```
git reflog --date=iso | head -10
# reset to safe state
git reset --hard <sha>
```

---

5 · Branching Strategy for Research Experiments

Naming Convention

```
main
feat/<topic>-<short-slug>
exp/<paper-acronym>/<experiment-id>
fix/<issue-id>
rel/v<major>.<minor>
```

Lightweight Experimental Loops

```
# Start hypothesis H1
git switch -c exp/GraphRAG/edge-drop-sweep
# Commit raw results
python scripts/run_sweep.py --cfg cfgs/edge_drop.yaml
# Stage & commit CSV + YAML
git add results/edge_drop_*.csv cfgs/edge_drop.yaml
```

Merge path: `exp/*` → `squash` into `feat/*` → PR to `main` once hypothesis validated.

---

6 · File Origin & Provenance Analysis

Research audits often ask "*Which commit produced Figure 3b?*".

Task	Command
First introduction of file	`git log --find-renames --diff-filter=A -- <file>`
All modifications (rename-aware)	`git log --follow -p -- <file>`
Pinpoint typo introduction	`git blame -L <start>,<end> <file>`
Trace function evolution	`git log -S 'def simulate_cfd' -p`
Cross-file symbol search	`git log -G 'sigma_layers\[' -p`

---

7 · Rich Log Traversal & Data Mining

7.1 Visual Topology

```
git log --graph --date=relative --pretty=format:'%C(auto)%h %d %s %C(blue)
(%ar)' --all | less -R
```

7.2 Temporal Slices & Statistics

```
# Commits per author last 8 weeks
git shortlog -sn --since="8 weeks ago"
# LOC churn per week
git log --since="1.year" --format=tformat: --numstat \
  | awk '{add+=$1; del+=$2} END{print add, del}'
```

7.3 JSON-ready Logs for Dashboards

```
git log --
pretty=format:'{"hash":"%H","author":"%an","date":"%ad","msg":"%s"},' \
         --since="2025-01-01" --reverse > logs/commits.json
```

---

8 · Productivity Aliases

```
[alias]
    # Clean merge-graph view
    lgg = log --graph --decorate --all --pretty=format:'%C(auto)%h %d %s'
    # Quick stash stack listing
    st  = stash list --pretty=format:'%gd · %C(yellow)%s%Creset'
    # Bring back last-stashed work
    pop = stash pop
    # Patch-mode add
    ia  = add -p
    # Copy file path of last commit (MacOS pbcopy)
    cfp = "!git diff-tree --no-commit-id --name-only -r HEAD | pbcopy"
```

---

9 · Recovery Playbook

Accident	Symptom	Command & Explanation
Overwrote branch with bad rebase	Missing commits	`git reflog`; `git branch rescue <sha>` then merge

Accident	Symptom	Command & Explanation
Force-push race condition	Remote rejects	`git pull --rebase --autostash` resolve ↔ `git push --force-with-lease`
Deleted untracked notebooks	Local files lost	`git fsck --lost-found` then inspect `.git/lost-found/other`
Corrupted object database	`fatal: bad object`	`git fsck`; `git repack -A -d -l`; last resort clone fresh

---

10 · Research-Specific Advice

1. **Datasets** ▶ Use `git-annex` or `dvc` for TB-scale assets; version pointers not blobs.
2. **Experiment IDs** ▶ Log hydra run dir + commit SHA into `wandb`/TensorBoard—enables cross-paper verification.
3. **LaTeX Repro** ▶ Store `latexmkrc` and `Dockerfile` for deterministic PDF builds.
4. **Licensing** ▶ Dual-license code (MIT) vs data (CC-BY-NC).
5. **Funding Acknowledgement** ▶ Draft in `docs/ack_funding.md` and link commit in grant reports.
6. **CI Minutes Conservation** ▶ Use self-hosted runner on university GPU node, triggered only on `main` & `rel/*`.

---

11 · Pre-Merge Checklist ("Green Bar Gate")

☑ All **hooks** pass (< 40 s total).
☑ `git diff --stat origin/main..HEAD` reviewed line-by-line.
☑ Messages follow **Conventional Commits**; plural imperative mood.
☑ **Changelog** updated via `cz bump` or manual.
☑ No data leaks: `detect-secrets scan --baseline .secrets.baseline`.
☑ At least **one peer review** & CI green.
☑ GPG signature verified on remote (`Verified`).

---

12 · Continuous Integration Template

```
# .github/workflows/ci.yaml
name: CI
on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
```

```
          fetch-depth: 0
      - uses: actions/setup-python@v5
        with:
          python-version: '3.12'
      - name: Install deps
        run: |
          sudo apt-get update && sudo apt-get install graphviz -y
          pip install -r requirements.txt
      - name: Lint & Typecheck
        run: |
          ruff check .
          mypy src/ --strict
      - name: Unit Tests
        run: pytest -q
      - name: Build Docs
        run: mkdocs build
```

---

13 · Worktrees, Sparse Checkout, and Partial Clones

13.1 Worktrees for Parallel Hotfixes

```
git worktree add ../bugfix-correct-load main
(cd ../bugfix-correct-load && <edit>)
```

13.2 Sparse Checkout for Notebooks Only

```
git sparse-checkout init --cone
git sparse-checkout set notebooks/
```

13.3 Blob-less Clone

```
git clone --filter=blob:none --depth=1 git@github.com:lab/monorepo.git
```

Cuts initial clone time on HPC nodes by > 90 %.

---

14 · Release & Archival Workflow

1. Create release branch `rel/v1.2`.
2. Update `CHANGELOG.md` + `version.py`.
3. Tag annotated: `git tag -a v1.2.0 -m "Release 1.2.0 – adds GNN-PDE coupling"`.
4. GitHub Actions generates wheels, Docker images, Zenodo DOI upload.
5. Merge `rel/v1.2` back into `main` **and** `develop` (if used).
6. Archive model checkpoints in `s3://lab-archive/monorepo/v1.2/` with tag SHA.

15 · Further Reading & Learning Paths

- Chacon & Straub, *Pro Git* (2023).
- Wlodzimierz Gajda, *Git Internals—Plumbing & Porcelain*.
- Jared Forsyth, *Conventional Commits Deep Dive* (YouTube).
- IEEE-TSE 2024: *Reproducibility Badges—A Systematic Review*.
- Software Heritage Archive—citing code artifacts.
- Zenodo + GitHub integration guide for archiving.

Happy version-controlling, and may your HEAD always be detached only by choice!