Data Use Academy and Consulting (DUAC)

DUAC · FREE GUIDE

Version Control with Git for Data Analysts

Track every change to your queries, scripts and reports, see exactly what changed, and go back safely when something breaks.

Version control · Git and GitHub

What you will learn

PARTS 1 AND 2

Set up and save well

A project Git can track, kept free of passwords and private data, and saved in small, clear steps.

PARTS 3 AND 4

See and undo changes

Find out what changed and who changed it, then go back to any earlier version without losing work.

PARTS 5 TO 7

Work as a team

Branches and reviews before changes reach the main version, plus Excel and Power BI files.

Before you start

You need Git installed and a free GitHub account. If you have neither yet, our guide **Build Your Data Portfolio on GitHub** at `du-ac.com/github-guide` sets up both: Part 1 for the account, Part 6 for Git.

Every example uses one project: a monthly sales report with a SQL query, an R script and a Power BI report. Type the commands in Git Bash (Windows) or Terminal (Mac), inside the project folder.

The words you will meet

Word	What it means
Repository	A project folder that Git tracks, on your computer and on GitHub.
Commit	A saved point in the history, with a message saying what changed and why.
Branch	A separate line of work, so changes do not touch the main version.
Pull request	A request on GitHub to bring a branch's changes into the main version.

Part 1 Set up a project Git can track

Decide what Git should track before your first commit. Keeping something out from the start is far easier than removing it later.

1. On GitHub, create a repository named `monthly-sales-report` with a README, then copy its address from the green **Code** button.
2. Download it to your computer and move into its folder:

```
git clone https://github.com/yourname/monthly-sales-report.git
cd monthly-sales-report
```

3. Create folders inside it so every project looks the same:

```
monthly-sales-report/
  README.md             what the project does, and where the data lives
  .gitignore            what Git must never track (next step)
  queries/              SQL files, such as monthly_sales.sql
  scripts/              R or Python, such as clean_sales.R
  reports/              the Power BI report, saved as a project (Part 6)
  data/raw/             raw data files, kept out of Git
```

4. In the project folder, create a plain text file named `.gitignore` containing:

```
# Data that is large, private or can be downloaded again
data/raw/
# Settings files that hold passwords and keys
.env
*credentials*
# Temporary files from Excel, R and Python
~$*
.Rhistory
.RData
__pycache__/
```

5. Save the setup as your first commit, then send it to GitHub:

```
git add .gitignore README.md
git commit -m "Set up project folders and .gitignore"
git push
```

WATCH OUT · Ignoring only works before a file is committed

If a data file was committed before you listed it in `.gitignore`, Git keeps tracking it. Run `git rm --cached data/raw/sales.csv` to stop tracking it without deleting your copy, then commit.

Part 2 Save your work in small steps

A commit is a saved point you can return to. Small commits with clear messages are what make the history useful later.

The save routine

```
git status                          # what has changed since my last commit?
git add queries/monthly_sales.sql   # choose what goes into this commit
git commit -m "Exclude test transactions from revenue"
git push                            # send it to GitHub
```

`git add .` stages every change in the folder at once. It is quicker, but run `git status` first, so nothing unexpected, such as a data file or a password, slips in.

Messages that help later

In six months, the message is all you will have to explain why a number changed. Start with a verb, say what changed and why, and keep it to one line.

Instead of	Write
`update`	`Exclude test transactions from revenue`
`fixed stuff`	`Count walk-in sales, which the join was dropping`
`final version`	`Add monthly totals by store for the Q3 report`
`changes`	`Use 2025 population estimates for rates`

One change per commit

- **Commit when one piece of work is done**, such as a new filter, a corrected join or a new chart, not once at the end of the day.
- **Keep unrelated changes apart.** A data fix and a colour change go in two commits, so either can be undone alone.
- **Push at least once a day**, so GitHub always has your latest work if your laptop fails.

TIP · Prefer buttons?

GitHub Desktop (`desktop.github.com`) shows changed files on the left. Tick the ones to include, type the message, then select **Commit to main** and **Push origin**.

CHECKPOINT · Before you move on

Run `git log --oneline`. You should see your commits, newest first, each with a message you would understand in six months.

Part 3 Find out what changed

When a number in your report moves, Git can tell you exactly which change moved it, who made it and when.

See the history

```
git log --oneline
7c2e1a9 Exclude test transactions from revenue
e41b0d3 Use 2025 population estimates for rates
9a6f2c8 Add delay rate by location
1d03e77 Set up project folders and .gitignore
```

Each line is one commit: its short ID, then its message. You use the ID to look at or undo that commit. When a list is long, Git shows it one screen at a time: scroll with the arrow keys and press **q** to get back to the prompt.

See what changed

Command	Shows you
`git diff`	Changes you have made but not yet staged
`git diff --staged`	Changes staged for your next commit
`git show 7c2e1a9`	Everything one commit changed, with its message and author
`git log -p queries/monthly_sales.sql`	The full story of one file, change by change

Reading a diff

```
queries/monthly_sales.sql
SELECT store_id, SUM(net_sales) AS revenue
FROM sales
-WHERE sale_date >= '2025-01-01'
+WHERE sale_date >= '2025-01-01'
+  AND payment_method <> 'TEST'
GROUP BY store_id;
```

Lines starting with **-** were removed and lines starting with **+** were added. Everything else is unchanged and only shown for context. Here, test transactions were taken out of revenue, which explains why the total fell.

The same, on GitHub

- Open any file, then select **History** to see every commit that changed it.
- Select **Blame** to see who last changed each line, and in which commit.
- Compare any two commits by adding `/compare/e41b0d3...7c2e1a9` to the repository's address.

Part 4 Go back safely

Every commit is kept, so a mistake can almost always be undone. Pick the row that matches your situation. The last column tells you if it is safe once your work is on GitHub.

I want to	Command	What happens	Safe after you push?
Throw away my unsaved edits to one file	`git restore queries/monthly_sales.sql`	The file goes back to its last commit. Your edits are gone.	**Yes**
Take a file out of the next commit	`git restore --staged queries/monthly_sales.sql`	Your edits stay, but the file is no longer staged.	**Yes**
Fix the message of my last commit	`git commit --amend -m "Better message"`	Replaces your last commit with a corrected one.	**No: only before you push**
Undo a whole commit	`git revert 7c2e1a9`	Adds a new commit that reverses it. The history stays.	**Yes: safest for shared work**
Look at an old version of a file	`git show e41b0d3:queries/monthly_sales.sql`	Prints that version. Nothing changes.	**Yes**
Bring back an old version of one file	`git restore --source=e41b0d3 queries/monthly_sales.sql`	Replaces the file with that version. Commit it to keep it.	**Yes**
See the whole project as it was	`git switch --detach e41b0d3` then `git switch main`	Opens every file as it was, read-only. Switch back when done.	**Yes**

Undo a mistake already on GitHub

```
git log --oneline       # find the commit
git show 7c2e1a9        # check what it did
git revert 7c2e1a9      # write the opposite
git push
```

Revert never deletes anything. The mistake and its correction both stay in the history, which is exactly what an auditor or a colleague needs to see.

WATCH OUT · Leave these alone for now

`git reset --hard` and `git push --force` rewrite history. Used wrongly, they delete work, yours or a colleague's. Revert does the same job safely.

TIP · Not sure? Look first

`git status`, `git log` and `git show` never change anything, so run them as often as you like.

Part 5 Work with others: branches and reviews

The `main` branch is the version everyone trusts. Make changes on a branch of your own, and bring them into `main` only after someone has checked them.

1. Start from the latest version: `git switch main`, then `git pull`.
2. Create a branch for your change and move onto it: `git switch -c add-delay-rate`.
3. Commit as usual. The first time you push the branch, run `git push -u origin add-delay-rate`.
4. On GitHub, select **Compare & pull request**. Say what changed, why, and how you checked the numbers, then select **Create pull request**.
5. A colleague reads the **Files changed** tab and comments. Once you agree, select **Merge pull request**.
6. Back on your computer, run `git switch main` and `git pull`, then delete the branch with `git branch -d add-delay-rate`.

> **TIP · What to check in a review**
>
> Does the change move any reported number? If so, is the reason in the pull request, and was the new figure checked against a known total? A review that asks these questions catches most errors before a report does.

When two changes collide

If you and a colleague change the same lines, Git cannot choose for you. It marks the clash in the file, called a merge conflict:

```
<<<<<<< HEAD
WHERE region = 'North Central'
=======
WHERE region IN ('North Central', 'North West')
>>>>>>> add-north-west
```

1. Open the file. The top half is your version, and the bottom half is the incoming one.
2. Keep the lines that are right, which may mean combining both, and delete the three marker lines.
3. Save the file, then run `git add queries/monthly_sales.sql` and `git commit` to finish.

Conflicts are normal and rarely serious. Pulling often and keeping branches short keeps them small.

Part 6 Excel, Power BI and data files

Git keeps every version of every file. What it cannot always do is show you what changed inside one.

File	Can Git show the changes?	What to do
SQL, R, Python, Markdown	**Yes, line by line**	Commit as normal. This is where Git helps most.
Small CSV files	**Yes, line by line**	Fine for lookup tables and small samples.
Excel workbooks	**No, but every version is kept**	Say what changed in the commit message. Keep important logic, such as queries, in text files beside the workbook.
Power BI `.pbix`	**No**	Save the report as a Power BI project instead.
Power BI `.pbip`	**Yes, as text files**	Commit the project's folders as normal.
Large or private data	**Never commit it**	Keep it in a shared drive or a database, and write down where in the README.

Save a Power BI report as a project

1. Open the report in Power BI Desktop and select **File** › **Save as**.
2. Set the file type to **Power BI project files (*.pbip)** and save it in your `reports/` folder.
3. Commit the new files. Power BI Desktop adds its own `.gitignore`, which keeps its local data cache out of Git.

A project holds the report's design and data model, not the data itself. Anyone who opens it refreshes the data with their own access, which keeps the data where it belongs.

Part 7 Habits that keep you safe

Habit	Done
Pull before you start work, so you build on the latest version.	☐
One change per commit, with a message that says why.	☐
Check `git status` before every `git add .`	☐
No passwords, keys or personal data in any commit.	☐
Use a branch and a review for anything that changes a reported number.	☐
Push at the end of every working day.	☐

Common problems and fixes

What you see	What it means	What to do
"nothing to commit, working tree clean"	Everything is already saved	Nothing. Make a change first.
"Changes not staged for commit"	You edited files but have not added them	Run `git add` on the files, then commit
"Your branch is behind 'origin/main'"	GitHub has commits your computer does not	Run `git pull`
Push refused: "fetch first"	Someone pushed before you did	Run `git pull`, then `git push` again
"CONFLICT (content): Merge conflict"	Two changes touched the same lines	Fix the file as shown in Part 5
"You are in 'detached HEAD' state"	You opened an old commit directly	Run `git switch main` to get back
A file you ignored is still tracked	It was committed before you added it to `.gitignore`	Run `git rm --cached` on it, then commit
A strange editor opened and you are stuck	You ran `git commit` without `-m`	Press **Esc**, type `:wq` and press **Enter**. Next time, add `-m "message"`
Long output and no prompt	Git is showing one screen at a time	Press **q**
You committed a password	It stays in the history even after you delete it	Change the password at once, make the repository private, and get help to clean the history

TIP · Keep learning

For why version control matters and how it fits into everyday analysis, read **Version Control for Data Analysts: How Git Keeps Your Work Safe** on the DUAC blog at `du-ac.com/blog`. If your work is not on GitHub yet, start with our free guide at `du-ac.com/github-guide`.