Git Bash: Step-by-Step Guide & Command Cheat Sheet

A practical, hands-on walkthrough to go from zero to confident with Git **using Git Bash on Windows**. Each section has copy-pasteable commands, explanations, and common fixes.

---

0) What is Git Bash?

Git Bash is a terminal that ships with *Git for Windows*. It gives you Unix-like commands (ls, pwd, cat) and the full Git CLI (git …). You'll use it to: - Track changes to code and docs - Collaborate via GitHub/GitLab/Bitbucket - Branch, merge, and undo safely

> Tip: Open Git Bash by searching "Git Bash" in Start Menu. Right-click a folder → **Git Bash Here** to start in that folder.

---

1) Install & First-Time Setup

1) **Install Git for Windows** - Download and install from the official site. During setup, safe defaults are: - **Default editor**: *Use Visual Studio Code* (or Vim/Notepad++) - **Adjust PATH**: *Git from the command line and also from 3rd-party software* - **Line endings**: choose **Checkout as-is, commit as-is** *(advanced)* or see step 1.3 below - **Terminal**: Use MinTTY (Git Bash)

2) **Verify installation**

```
git --version
```

3) **Identify yourself (required)**

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
git config --global init.defaultBranch main
```

4) **Quality of life (recommended)**

```
git config --global color.ui auto
# Use VS Code as editor (if installed)
git config --global core.editor "code --wait"
```

```
# Use Git Credential Manager (stores HTTPS creds securely on Windows)
git config --global credential.helper manager
```

5) **Line endings (Windows vs. Linux/macOS)** Pick ONE policy and stick to it across your machines: - Keep repo files as LF (common for Node/Angular/.NET cross-platform):

```
git config --global core.autocrlf input   # commit LF, don't modify checkout
```

- Convert LF↔CRLF automatically on Windows checkout/commit (classic Windows workflow):

```
git config --global core.autocrlf true
```

---

2) HTTPS vs SSH (how you authenticate)

HTTPS is simplest (uses Git Credential Manager). **SSH** avoids typing passwords and is preferred for teams.

Setup SSH keys (recommended)

```
# 2.1 Generate a key (press Enter for defaults, optionally add a passphrase)
ssh-keygen -t ed25519 -C "you@example.com"

# 2.2 Start the SSH agent & add your key
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519

# 2.3 Copy the public key to clipboard and add it to GitHub → Settings → SSH
keys
clip < ~/.ssh/id_ed25519.pub

# 2.4 Test the connection
ssh -T git@github.com
```

> If you see a "successfully authenticated" message, you're good.

---

3) Create a brand-new project & push to GitHub

```
# 3.1 Make a project folder and initialize Git
mkdir MyProject && cd MyProject
git init
```

```
# 3.2 Add some files (example README)
echo "# MyProject" > README.md

# 3.3 Stage & commit your first snapshot
git add .
git commit -m "chore: initial commit"

# 3.4 Create an empty repo on GitHub (no README/License to avoid conflicts)
# 3.5 Link local repo to GitHub and push main
# Use the SSH or HTTPS URL from GitHub
git remote add origin git@github.com:USERNAME/MyProject.git

git push -u origin main
```

> `-u` sets *upstream* so future `git push` and `git pull` know the default remote/branch.

---

4) Clone an existing repository

```
# SSH (recommended)
git clone git@github.com:OWNER/Repo.git
# or HTTPS
git clone https://github.com/OWNER/Repo.git

cd Repo
git remote -v   # show remotes
```

---

5) Daily workflow (single-branch)

```
# 5.1 Pull latest before you start (rebase creates a tidy linear history)
git pull --rebase

# 5.2 Check what changed
git status

# 5.3 Stage changes (all or specific files)
git add -A            # everything
git add path/file.cs  # specific

# 5.4 Commit with a clear message
git commit -m "feat: export to CSV from report page"
```

```
# 5.5 Push your work
git push
```

---

6) Branch-based feature workflow (recommended for teams)

```
# 6.1 Create & switch to a feature branch
git switch -c feature/login

# 6.2 Work normally: edit → add → commit (repeat)
# ...

# 6.3 Publish the branch & open a Pull Request on GitHub
git push -u origin feature/login

# 6.4 After PR is merged, sync main locally
git switch main
git pull --ff-only

# 6.5 Clean up merged branches
git branch -d feature/login                  # delete local
git push origin --delete feature/login       # delete remote
```

Useful branch commands

```
git branch                  # list local branches
git branch -a               # include remotes
git switch <name>           # move between branches
# (older) git checkout -b newbranch   # create + switch
```

---

7) See history & differences

```
# Compact, visual history (alias suggested in §12)
git log --oneline --graph --decorate --all

# Show file differences
git diff                   # unstaged changes
git diff --staged          # compare staged vs last commit
git show <commit>          # what changed in a specific commit
```

```
# Who changed each line
git blame path/file.ts
```

---

8) Undo & fix safely

Golden rule: prefer *revert* on shared branches (history safe). Use *reset* only on your own local commits.

```
# 8.1 Edit the last commit message or include forgotten files
git commit --amend -m "fix: correct typo in README"

# 8.2 Unstage but keep changes
git restore --staged path/file.cs

# 8.3 Discard local changes to a file (careful!)
git restore path/file.cs

# 8.4 Revert a bad commit (makes a new commit that undoes it)
git revert <commit>

# 8.5 Reset to a previous commit (moves branch pointer)
# soft: keep all changes staged
git reset --soft HEAD~1
# mixed (default): keep changes, unstage them
git reset HEAD~1
# hard: discard everything (DANGEROUS)
git reset --hard HEAD~1

# 8.6 Recover with the reflog (time machine)
git reflog        # find the lost commit/HEAD, then
git reset --hard <reflog-id>
```

---

9) Stash work in progress (WIP)

```
git stash push -m "WIP: partial login UI"
git stash list
# bring it back (and remove from stash)
git stash pop
# or apply without removing from stash
git stash apply stash@{2}
# remove a stash entry
git stash drop stash@{0}
```

10) Remotes: origin vs upstream

Use this when you fork a repo.

```
# origin = your fork, upstream = original project
git remote -v
git remote add upstream git@github.com:ORIGINAL/Repo.git

# Sync your main with upstream
git fetch upstream
git switch main
git rebase upstream/main

git push --force-with-lease  # only to your fork, after a rebase
```

Other remote maintenance:

```
git remote rename origin origin-github
git remote remove old-remote
```

11) Tags & releases

```
git tag                    # list tags
git tag -a v1.0.0 -m "Initial release"
git push origin v1.0.0
```

12) Handy aliases (type less)

```
git config --global alias.st "status -sb"
git config --global alias.co "switch"
git config --global alias.br "branch"
git config --global alias.cm "commit -m"
git config --global alias.lg "log --oneline --graph --decorate --all"
```

Use like: `git st`, `git co feature/x`, `git lg`.

13) .gitignore (keep noise out of your repo)

Create a `.gitignore` at the repo root. Examples:

```
# Node/Angular
node_modules/
dist/

# .NET / Visual Studio
bin/
obj/
.vs/
*.user

# OS/editor junk
.DS_Store
Thumbs.db
*.log
```

If you already committed something that should be ignored:

```
git rm -r --cached node_modules
```

> Large binaries? Consider **Git LFS** (Large File Storage) for assets: psd, zip, mp4. (Install LFS → `git lfs track "*.mp4"` → commit the `.gitattributes` it creates.)

---

14) Merge conflicts (how to resolve)

1) You'll see markers in files like:

```
<<<<<<< HEAD
Your changes
=======
Incoming changes
>>>>>>> origin/main
```

2) Edit the file to keep what you want. 3) Mark as resolved and continue:

```
git add path/conflicted-file
# if merging
git merge --continue
```

```
# if rebasing
git rebase --continue
```

4) If you get stuck:

```
git merge --abort   # or
git rebase --abort
```

---

15) Common errors & quick fixes

- **"fatal: not a git repository"** → You're not inside a repo. Run `git status`. If needed: `git init`.
- **"Updates were rejected because the remote contains work…"** → `git pull --rebase` first, resolve conflicts, then `git push`.
- **"Permission denied (publickey)"** → SSH key not added. Re-run §2, check `ssh-add -l`.
- **LF/CRLF warnings** → Set `core.autocrlf` in §1.5 and/or use a `.gitattributes` file to normalize line endings.
- **Pushed the wrong branch** → `git push origin --delete wrong-branch` then `git push -u origin correct-branch`.

---

16) Advanced (optional but useful)

```
# Rebase your feature branch onto updated main (clean history)
git switch feature/login
git fetch origin
git rebase origin/main

# Pick a commit from another branch
git cherry-pick <commit>

# Squash multiple commits interactively (local feature branches only)
git rebase -i HEAD~5
# change 'pick' → 'squash' for the commits you want to combine

# Rename a branch
git branch -m old-name new-name
```

---

17) Practice lab (10 minutes)

1) `mkdir lab && cd lab && git init` 2) `echo "hello" > a.txt && git add . && git commit -m "add a"` 3) `git switch -c feature/x && echo "x" >> a.txt && git add . && git commit -m "feat: x"` 4) `git switch main && echo "main" >> a.txt && git commit -am "feat: main change"` 5) `git merge feature/x` → resolve conflict → `git add a.txt && git merge --continue` 6) `git log --oneline --graph --decorate --all` to review

---

18) Quick reference (most-used commands)

```
# Setup
git config --global user.name "Name"
git config --global user.email "you@example.com"

# New repo → GitHub
git init && git add . && git commit -m "init"
git remote add origin <url>
git push -u origin main

# Daily
git pull --rebase
git status
git add -A
git commit -m "message"
git push

# Branching
git switch -c feature/x
git push -u origin feature/x

git switch main
git pull --ff-only

# Inspect
git lg
git diff
git show <commit>

# Undo
git commit --amend
git revert <commit>
# (local only) git reset --hard <commit>

# Stash
```

```
git stash push -m "msg"
git stash pop
```

Final tips

- Commit small, logical changes with clear messages (use conventional commits if you can: `feat:`, `fix:`, `docs:` …).
- Pull before you start work; push when you're done.
- Prefer branches + PRs for anything non-trivial.
- When in doubt, `git status`, `git log`, and `git reflog` are your best friends.

Happy shipping! 🚀