# Git — Practical Study Sheet

**Goal.** A zero-fluff, copy&paste guide that you can run end-to-end: setup, daily workflow, remote sync, authentication, branching choices, merge strategies, and a hands-on lab.

## 1) First-time setup (one-off)

```
git --version
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
git config --global init.defaultBranch main
# Optional quality-of-life
git config --global pull.rebase false # or true if you prefer linear history
git config --global fetch.prune true
git config --global color.ui auto
```

Initialize or clone:

```
git init
git clone <url>
```

Remotes:

```
git remote -v
git remote add origin <url>
git remote set-url origin <new-url>
git remote remove origin
```

## 2) Daily local workflow

```
git status
git add <file> # or: git add .
git commit -m "message"
git log --oneline --graph --decorate
```

Branch and switch:

```
git branch <name> # create branch
git branch # list branches
git checkout <name> # switch
# or modern:
git switch -c <name> # create+switch
git switch <name> # switch
```

## 3) Sync with remote

- **Fetch** to preview incoming changes without merging.
- **Pull** to fetch and integrate into your branch.
- **Push** to publish your local commits.

```
git fetch
git pull # fetch + merge (or rebase if configured)
git push origin main
git push -u origin <branch> # set upstream tracking the first time
```

## 4) Authentication (pick one)

### A. HTTPS with PAT (Personal Access Token)

1. Create a PAT (fine-grained or classic). Grant minimal scopes (e.g., repo).
2. Use a credential helper (recommended) or paste when prompted.
3. For demos only, embed PAT in URL (avoid in real projects):

```
# Demo only
git clone https://<PAT>@github.com/<user>/<repo>.git
```

Credential helper examples:

```
git config --global credential.helper manager-core # Windows
git config --global credential.helper osxkeychain # macOS
git config --global credential.helper store # Plaintext file (avoid)
```

### B. SSH keys (convenient & secure)

```
# Create key (with passphrase recommended)
ssh-keygen -t ed25519 -C "you@example.com"

# Copy pubkey to clipboard (Windows example)
clip < ~/.ssh/id_ed25519.pub

# Add the public key to GitHub: Settings -> SSH and GPG keys
# Then use the SSH URL:
git clone git@github.com:<user>/<repo>.git
```

## 5) Branching strategies (what & when)

**GitHub Flow**
Feature branch → Pull Request → main, deploy continuously. Simple and great for frequent releases.

**Git Flow / Release branches**
Long-lived branches (develop, release, hotfix). Useful for regulated or complex release trains.

**Trunk-based**
Very short-lived branches, frequent merges to main with robust CI.

## 6) Merge choices (how your history looks)

- **Merge commit** (default): preserves a merge point (commit with two parents).
- **Rebase**: replay feature commits onto target; linear history; changes commit SHAs.
- **Squash & merge**: combine all feature commits into a single commit on target.

Tiny ASCII previews (left-to-right history):

```
# Merge commit
*---*---*---M (main)
     \-----*---* (feature)

# Rebase (linear)
*---*---*---*---* (main after rebased feature)

# Squash & merge
*---*---*---S (main, one squashed commit)
     \-----*---* (feature branch history stays on branch)
```

Common commands:

```
# Merge commit
git checkout main
git pull
git merge feature/xyz

# Rebase feature onto main (linear history)
git checkout feature/xyz
git fetch origin
git rebase origin/main
# If conflicts: resolve, then
git rebase --continue

# Squash locally (interactive)
git checkout feature/xyz
git rebase -i origin/main # mark subsequent commits as "squash" or "fixup"
```

## 7) Hands-on lab (CLI + Desktop + GUI)

### Part A: Create a repository (CLI)

1. Initialize and commit:

```
mkdir demo-git && cd demo-git
git init
echo "# Demo" > README.md
git add README.md
git commit -m "chore: initial commit"
```

2. Inspect history:

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

### Part B: Connect a remote & publish

```
git remote add origin <ssh-or-https-url>
git push -u origin main
```

### Part C: Auth drills

- **PAT (HTTPS)**: create PAT, set credential helper, clone via HTTPS.
- **SSH**: `ssh-keygen`, add public key, clone via SSH URL.

### Part D: Branch & merge exercises

1. Create three branches: `merge-commit`, `rebase`, `squash`.

```
git switch -c merge-commit
echo "a" >> notes.txt && git add notes.txt && git commit -m "add a"
git switch -c rebase main
echo "b" >> notes.txt && git add notes.txt && git commit -m "add b"
git switch -c squash main
echo "c1" >> notes.txt && git add notes.txt && git commit -m "add c1"
echo "c2" >> notes.txt && git add notes.txt && git commit -m "add c2"
```

2. Update `main` to diverge:

```
git switch main
echo "main" >> notes.txt && git add notes.txt && git commit -m "main edit"
```

3. Merge commit path:

```
git merge merge-commit
```

4. Rebase path:

```
git switch rebase
git rebase main
git switch main
git merge --ff-only rebase
```

5. Squash path (local):

```
git switch squash
git rebase -i main # mark second commit as "squash" or "fixup"
git switch main
git merge --ff-only squash
```

6. Inspect results:

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

## 8) Quick troubleshooting (fast fixes)

**`fatal: refusing to merge unrelated histories`**
Add `-allow-unrelated-histories` or re-create remote to match local.

**Detached HEAD**
You checked out a commit, not a branch. Create a branch: `git switch -c fix/my-branch`.

**Undo last commit (keep changes staged)**
`git reset -soft HEAD~1`

**Undo last commit (keep changes unstaged)**
`git reset -mixed HEAD~1`

**Discard local changes to a file**
`git checkout - <file>` (old) or `git restore <file>` (new).

**Overwrite local with remote**
`git fetch; git reset -hard origin/main` (destructive).

**Clean untracked files**
`git clean -fd` (destructive: removes files/dirs not tracked).

## 9) Mental model (TL;DR)

- **Stage** (`git add`) picks *exactly* what goes into the next snapshot.
- **Commit** (`git commit`) freezes that snapshot with a message.
- **Branch** is a movable pointer to a sequence of commits.
- **Fetch/Pull/Push** synchronize your local refs with remotes.
- Pick a merge strategy that matches your team's review and release habits.

*Tip:* save these dotfiles for consistent behavior across machines:

```
# ~/.gitconfig (excerpt)
[pull]
    rebase = false
[fetch]
    prune = true
[color]
    ui = auto
```