Git Reference for Hackathon 2026

Git + VS Code Cheat Sheet (Windows, 30-minute workshop)

1) One-time setup (do this once per computer)

Install Git

- Install **Git for Windows** (includes “Git Bash”).
- Verify in VS Code Terminal (or Git Bash):

```
git --version
```

Configure your identity (required for commits)

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

Helpful defaults

```
git config --global init.defaultBranch main
```

---

2) The mental model (memorize this)

Working Directory → Staging Area → Commit History

- **Edit files** (working directory)
- **Stage** what you want included
- **Commit** = a saved checkpoint (local)
- **Push** = publish commits to a remote (e.g., GitHub)
- **Pull** = get remote updates and integrate into your branch

---

3) Your “daily loop” (do this constantly)

In **VS Code → Source Control**:

1. Check changes (diff)
2. **Stage** (+) the right files
3. Write a **meaningful message**
4. **Commit**
5. **Push** occasionally (so your work is backed up and shareable)

Command-line equivalents

```
git status
git add <file>        # or: git add .
git commit -m "Add X"
git push
```

---

4) Starting work: clone vs init

If the repo already exists on a remote (common in classes/teams):

```
git clone <URL>
```

If you are starting a brand-new project locally:

```
git init
```

(Then later create a remote repo and connect it.)

---

5) Branch workflow (teams should use this)

Rule: Do not commit directly to `main` in a team project.

Create a branch

- VS Code: click the branch name in the bottom-left → “Create new branch”

```
git switch -c feature/short-description
```

Commit on the branch, then push it:

```
git push -u origin feature/short-description
```

Open a Pull Request (PR) on GitHub

- PR = request to merge your branch into `main`
- (GitLab calls these “merge requests”: GitLab)

Merge only when:

- the change is coherent
- reviews are done
- (if applicable) tests/checks pass
- `main` stays working

---

6) Keeping up to date (avoid “surprise conflicts”)

Safe habit

```
git fetch
```

Then decide what to do.

Typical update

```
git pull
```

Note: pull may merge into your branch.

---

7) Fixing mistakes (high-value “rescue” commands)

See what’s going on (always first)

```
git status
git log --oneline --graph --decorate
git diff
git diff --staged
```

Unstage a file (keep edits)

```
git restore --staged <file>
```

Discard local edits to a file (be careful)

```
git restore <file>
```

Stash: temporarily shelve uncommitted work

```
git stash
git stash pop
```

Use when you need a clean working directory to switch branches.

---

8) Commit messages students can follow

Good:

- `Add input validation to signup form`
- `Fix crash when file is missing`
- `Refactor parsing logic for readability`

Avoid:

- `stuff`
- `final`

- `pls work`

Practice: one commit = one coherent change.

---

9) Team hygiene (quick rules)

- Protect `main`: require PRs + at least 1 approval + resolve conversations
- Add a `.gitignore` early (don’t commit build output, IDE junk, secrets)
- Push work regularly (your laptop is not a backup)

---

10) Minimal command set to memorize

```
git status
git add .
git commit -m "Message"
git pull
git push
git switch -c <branch>
git switch main
```