Git Cheat Sheet

1 Git & Version Control Essentials

Why VCS?
Track file changes over time, collaborate seamlessly, and maintain a complete history of your project.

Types of VCS:

- **Centralized:** (e.g., SVN, TFS) – Single point of failure, requires constant server access.
- **Distributed:** (e.g., Git, Mercurial) – Every user holds a full copy; enables offline work.

Tooling Tips:

- **CLI:** Offers full control and conceptual clarity.
- **GUIs & IDE Extensions:** GitKraken, Sourcetree, VS Code (GitLens) – handy for visual diffing but may hide details.

2 Setup & Configuration

Installing Git:

```
# Download from: https://git-scm.com/downloads
git --version  % Verify installation
```

Initial Configuration:

```
git config --global user.name "your_name"
git config --global user.email "your_email@example.com"
git config --global core.editor "code --wait"  % Set VS Code as default editor
git config --global -e                          % Edit global config
```

Managing Line Endings:

```
# Windows users:
git config --global core.autocrlf true
# Mac/Linux users (if needed):
git config --global core.autocrlf input
```

Documentation: https://git-scm.com/docs

3 Basic Workflow & Snapshots

Initializing & Creating Snapshots:

```
git init                        % Start a new repository
git add <file> or git add .     % Stage changes
git commit -m "message"         % Commit staged changes
```

Common Commands:

```
git status                      % Check working/staging status
git diff                        % View unstaged changes
git diff --staged               % View staged changes
git log --oneline               % Compact commit history
```

4 Reviewing Changes & History

History & Inspection:

```
git log                       % Full commit history
git log --oneline --graph     % Visual commit tree
git show <commit_hash>        % Details of a specific commit
git blame <file>              % Identify who last changed each line
```

Search Options:

```
git log --grep="term"           % Search commit messages
git log -S"code"                % Find commits modifying specific code
```

5 Branching, Merging & Advanced Operations

5.1 Branching and Stashing

```
git branch <branch_name>       % Create branch
git switch -c <branch_name>    % Create and switch to branch
git branch -m old new          % Rename branch
git branch -d <branch_name>    % Delete branch

git stash push -m "message"    % Stash changes
git stash list                 % List stashes
git stash pop                  % Apply stash
```

5.2 Merging & Rebasing

```
git merge <branch>             % Merge branch into current
git merge --squash <branch>    % Squash merge (combine commits)
git merge --abort              % Abort merge in progress

git rebase master              % Rebase current branch onto master
git rebase --continue          % Continue after conflict resolution
git rebase --abort             % Cancel rebase
```

5.3 Advanced Commands

```
git commit --amend           % Amend last commit
git cherry-pick <commit_hash>  % Apply specific commit from another branch
git bisect start             % Begin binary search for faulty commit
git bisect good/bad <commit>   % Mark commits during bisect
git bisect reset             % End bisect mode
git tag v1.0                 % Tag the current commit
```

6 Collaboration & Remotes

Cloning & Remote Setup:

```
git clone <url>               % Clone a remote repository
git remote -v                 % List remotes
git remote add upstream <url> % Add new remote (e.g., original repo)
```

Synchronizing Changes:

```
git fetch origin              % Fetch remote updates without merging
git pull                      % Fetch and merge remote changes
git push                      % Push local commits to remote
git push origin <branch>      % Push specific branch
git push --set-upstream origin <branch>  % Set upstream for new branch
```

Tags & Cleanup:

```
git push origin <tag>        % Push a tag to remote
git push origin --delete <branch>  % Delete remote branch
git remote prune origin      % Clean up stale remote branches
```

7 Best Practices & Tips

- **Commit Often:** Keep commits self-contained and logical.
- **Meaningful Messages:** Use clear, concise commit messages.
- **Review Before Merging:** Use diff tools (e.g., VS Code, KDiff3) to inspect changes.
- **Backup with Branches:** Use feature branches and stashing to avoid losing work.
- **CLI First:** Master the command line before relying solely on GUIs.

Issue Tracking:
Use GitHub/GitLab Issues and Milestones for bug tracking and feature requests.