The Comprehensive Git Cheat Sheet

For Systems Programmers & CLI Power Users

1. Configuration & Setup

Set up your identity and preferences once per machine.

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

# Editor (Essential for commit messages and interactive rebases)
git config --global core.editor "nvim"  # or "vim"

# Initialization
git init                 # Initialize a repo in current directory
git clone <url>          # Clone a remote repo
git clone --depth 1 <url> # Shallow clone (latest snapshot only). Great for large codebases
like Linux Kernel.


git status               # detailed status
git status -s            # short/concise status

# Staging Files
git add <file>           # Stage a specific file
git add .                # Stage all changes in current dir
git add -p               # Interactive Patch Mode. Decide hunk-by-hunk. Essential for clean
commits.

# Committing
git commit -m "msg"      # Commit with inline message
git commit               # Opens $EDITOR for a detailed message
git commit --amend       # Modify the PREVIOUS commit (change msg or add forgotten files).
                         # WARNING: Do not amend commits already pushed to a shared remote.


# Listing
git branch               # List local branches
git branch -a            # List local and remote branches

# Creating & Switching
git switch <name>        # Switch to an existing branch (modern alternative to checkout)
git switch -c <name>     # Create AND switch to a new branch
git checkout <name>      # Old school switch
git checkout -b <name>   # Old school create & switch

# Merging
git merge <branch>       # Merge <branch> into your CURRENT branch

# Managing Connections
```

```
git remote -v                     # List remotes (verbose)
git remote add origin <url>       # Add a new remote named 'origin'
git remote set-url origin <url>   # Change the URL (e.g., HTTPS -> SSH)
git remote rename origin upstream # Rename a remote (useful when forking)
git remote remove <name>          # Remove a remote connection

# Syncing
git fetch                         # Download data from remote, but DO NOT modify files.
git pull                          # Fetch + Merge.
git push origin <branch>          # Upload local branch to remote.
git push -u origin <branch>       # Push and set "upstream" tracking.
git remote prune origin           # Delete local refs to branches that no longer exist on
remote.

# Logs
git log                           # Full history
git log --oneline                 # Compact history
git log --graph --oneline --all   # Visual ASCII graph of all branches
git log -p <file>                 # Show history of actual code changes for a file

# Diffing
git diff                          # Unstaged changes (Working Dir vs Index)
git diff --staged                 # Staged changes (Index vs Last Commit)
git diff HEAD~1                   # Compare current state vs 1 commit ago

# Auditing
git blame <file>                  # Show who last modified each line. Great for legacy C code.
```

```
# Logs
git log                           # Full history
git log --oneline                 # Compact history
git log --graph --oneline --all   # Visual ASCII graph of all branches
git log -p <file>                 # Show history of actual code changes for a file

# Diffing
git diff                          # Unstaged changes (Working Dir vs Index)
git diff --staged                 # Staged changes (Index vs Last Commit)
git diff HEAD~1                   # Compare current state vs 1 commit ago

# Auditing
git blame <file>                  # Show who last modified each line. Great for legacy C code.
```

```
git rm <file>           # Delete file from disk AND git
git rm --cached <file>  # Stop tracking file, but KEEP it on disk (fixes .gitignore mistakes)
git mv <old> <new>      # Rename/Move a file

# Rebasing (The linear history tool)
git rebase <branch>     # Replay your commits on top of <branch>
git rebase -i HEAD~3    # Interactive Rebase. Squash, Reword, Edit, or Drop the last 3
commits.

# Cherry-Picking
git cherry-pick <hash>  # Copy a specific commit from another branch to here.
```

```
# Restoring Files
git restore <file>           # Discard local changes (undo edits)
git restore --staged <file> # Unstage a file (keep edits, just remove from Index)

# Resetting (Time Travel)
git reset --soft HEAD~1      # Undo last commit, keep changes staged.
git reset --hard HEAD~1      # DESTROY last commit and changes. (Dangerous)

# The Safety Net
git reflog                   # Show the log of where HEAD has pointed (tracks
resets/checkouts).
git reset --hard HEAD@{n}    # Restore repo to a state seen in reflog.

# Stashing (Temporary Storage)
git stash                    # Save dirty changes to a stack
git stash push -m "wip-socket" # Stash with a name
git stash pop                # Apply stashed changes back
git stash list               # See all stashes


# Tagging
git tag -a v1.0 -m "Release"  # Create an annotated tag
git push origin --tags        # Push all tags to remote
git tag -d v1.0               # Delete local tag
git push origin --delete v1.0 # Delete remote tag

#Debugging
git bisect start              # Start binary search to find a bug
git bisect bad                # Mark current commit as broken
git bisect good <commit>      # Mark an older commit as working
# (Git will now checkout the middle commit for you to test)

#Maintenance
git clean -fd                 # Force remove untracked files/directories (build artifacts)
git archive --format=zip --output=code.zip master # Zip the code without .git folder
```