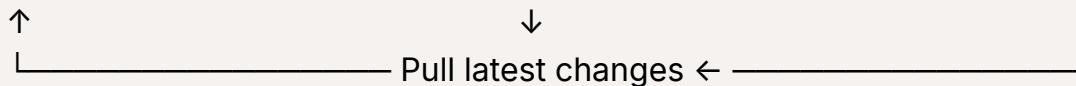


Git Cheatsheet

Development Cycle (The Daily Loop)

Write code → Test locally → Stage changes → Commit → Push to GitHub



Core Workflow

```
git status          # Check current state
git add <file> | git add .      # Stage changes
git commit -m "message"        # Save snapshot
git log --oneline --graph      # View history
git diff              # See unstaged changes
git diff --staged      # See staged changes
```

Setup & Config

```
git init            # Initialize repo
git clone <url>      # Copy remote repo
git config --global user.name "Name"
git config --global user.email "email"
git remote add origin <url>    # Link to remote
git remote -v        # View remotes
```

Branching

```
git branch          # List branches
git branch <name>    # Create branch
git switch <name>    # Switch branch
git switch -c <name> # Create + switch
```

git merge <branch>	# Merge into current
git branch -d <name>	# Delete branch
git rebase <branch>	# Rebase onto branch

Remote Sync

git fetch origin	# Download updates (no merge)
git pull	# Fetch + merge
git pull --rebase	# Fetch + rebase
git push	# Upload commits
git push -u origin <branch>	# Push + set upstream
git push --force-with-lease	# Safe force push

Undoing

git restore <file>	# Discard working changes
git restore --staged <file>	# Unstage file
git reset HEAD~1	# Undo commit (keep changes)
git reset --soft HEAD~1	# Undo commit (keep staged)
git reset --hard HEAD~1	# Undo commit (DELETE changes)
git commit --amend	# Fix last commit
git revert <hash>	# Undo commit (new commit)
git reflog	# View all actions (recovery)

Stashing

git stash	# Save work temporarily
git stash list	# View stashes
git stash pop	# Apply + delete stash
git stash apply	# Apply stash (keep it)
git stash drop	# Delete stash

Merge Conflicts

```
# 1. See conflicts
git status

# 2. Edit files, remove markers:
<<<<<< HEAD
current branch code
=====
incoming branch code
>>>>>> branch-name

# 3. Resolve
git add <resolved-file>
git commit -m "Resolve conflict"
```

Best Practices

Commit Messages

```
Good: "Fix memory leak in data loader"
Bad:  "fix" | "changes" | "update"

Format: <type>: <what changed>
Types: feat, fix, docs, refactor, test, chore
```

.gitignore Essentials

```
# Python
__pycache__/
*.pyc
.env
.venv/
*.egg-info/
```

```
# Data
*.csv
*.pkl
*.h5
data/raw/*
!data/sample.csv    # Keep this one

# Secrets
.env
*.key
config/secrets/

# OS/IDE
.DS_Store
.vscode/
.idea/
```

When to Branch

```
feature/*    → New features
bugfix/*     → Bug fixes
hotfix/*     → Urgent fixes
experiment/* → Experiments
```

Always branch for features
main = production-ready only

Decision Trees

Merge vs Rebase

Need exact history? → merge
Want clean history? → rebase

NEVER rebase shared branches
ALWAYS rebase local features

Undo Decision

Mistake not committed? → git restore
Committed locally? → git reset
Committed + pushed? → git revert
Everything broke? → git reflog

Emergency Fixes

Wrong Branch

```
git branch feature-x      # Create branch
git reset --hard HEAD~1   # Remove from current
git switch feature-x      # Check it's there
```

Lost Commits

```
git reflog                # Find commit
git switch -c recovery <hash> # Recover
```

Committed Secrets

```
git rm --cached secrets.env
git commit --amend
echo "secrets.env" >> .gitignore
# If pushed: Rotate secrets immediately
```

Quick Reference

Command	What It Does
<code>git status</code>	Current state (use constantly)
<code>git log --oneline</code>	Commit history
<code>git diff</code>	Unstaged changes
<code>git add .</code>	Stage everything
<code>git commit -m</code>	Save snapshot
<code>git push</code>	Upload to remote
<code>git pull</code>	Download + merge
<code>git switch -c</code>	Create branch
<code>git merge</code>	Combine branches
<code>git reflog</code>	Emergency recovery

Pro Tips

- Use `git status` before/after every command
- Commit small, commit often
- Pull before push
- Never force push to shared branches
- Test before commit
- Branches are free - use them