DevSheets

Git

Set up repos, commit changes, branch and sync with remotes.

01 Install & configure

```
brew install git                # macOS
winget install --id Git.Git -e  # Windows
sudo apt install git            # Ubuntu
git --version
git config --global user.name "Your Name"
git config --global user.email you@mail.com
git config --global init.defaultBranch main
git config --list               # check
```

Drop --global to set a value for this repo only.

02 Create & clone

```
git init                    # this folder
git clone <url>
git clone <url> my-dir
git clone -b develop <url>  # on develop
git clone --depth 1 <url>   # shallow
git clone --recursive <url> # + submodules
```

--depth 1 fetches only the latest commit: faster for big repos.

03 .gitignore

`node_modules/`	A directory
`*.log`	Every .log file
`!keep.log`	Exception: track it anyway
`/secret.txt`	Only in the repo root
`**/temp`	temp at any depth

```
git rm --cached .env        # stop tracking
git check-ignore -v <file>  # which rule?
```

Rules only affect untracked files. Commit .gitignore.

04 Status & diff

`git status`	Staged, modified, untracked
`git status -s`	Short format
`git diff`	Unstaged changes
`git diff --staged`	Staged changes (or --cached)
`git diff HEAD`	All changes since last commit
`git diff main`	Compare with a branch

Add a path to limit a diff to one file; -w ignores whitespace.

05 Stage & commit

```
git add <file>
git add .       # all, from this folder
git add -A      # all, whole repo
git add -p      # choose hunks
git commit -m "Add login form"
git commit -am "Fix typo"   # tracked files
git commit --amend -m "New message"
```

--amend rewrites the last commit (--no-edit keeps the message). Never amend a pushed commit.

06 Log & history

```
git log --oneline --graph --all
git log -n 5                # last 5
git log --author="Ada"
git log --grep="fix"        # messages
git log -S "getUser"        # code changes
git log -p <file>           # file history
git show <commit>
git blame <file>            # who wrote it
```

git log main..feature lists commits in feature that main lacks.

07 Branches

```
git branch                  # local
git branch -a               # with remote
git switch -c feature/login
git switch main
git branch -m new-name      # rename current
git branch -d feature/login  # if merged
git branch -D feature/login  # force
git push origin --delete feature/login
```

switch (Git 2.23+) replaces checkout -b and checkout <branch>. branch -vv shows tracking info.

08 Merging

```
git switch main
git merge feature           # into current
git merge --no-ff feature   # merge commit
git merge --ff-only feature  # or fail
git merge --squash feature  # then commit
git merge --abort           # back out
```

Fast-forward keeps history linear; --no-ff always records the merge. Preview incoming commits with git log main..feature.

09 Remotes, pull & push

```
git remote -v
git remote add origin <url>
git remote set-url origin <new-url>
git fetch --prune        # drop stale refs
git pull                 # fetch + merge
git pull --rebase        # fetch + rebase
git push -u origin main  # set upstream
git push --force-with-lease  # safer force
```

After push -u, plain git push and git pull use that branch.

Git

Undo mistakes, stash work, rewrite history and tag releases.

10 Discard & unstage

```
git restore <file>           # discard edits
git restore .                # every file
git restore --staged <file>  # unstage
git restore --source=<commit> <file>
git clean -n                 # preview
git clean -fd                # delete them
git clean -fdx               # + ignored
```

restore (Git 2.23+) replaces checkout -- <file> and reset HEAD <file>. -fdx also wipes node_modules.

11 git reset & reflog

`--soft HEAD~1`	Undo commit, keep staged
`--mixed HEAD~1`	Undo commit, keep unstaged
`--hard HEAD~1`	Undo commit, discard it all
`--hard <sha>`	Move the branch to <sha>

```
git reflog                     # HEAD log
git switch -c rescue HEAD@{1}  # recover
git reset --hard HEAD@{1}
```

--mixed is the default. Only reset unpushed work; the reflog keeps lost commits for at least 30 days.

12 Revert commits

```
git revert <commit>      # inverse commit
git revert HEAD          # undo the last one
git revert -m 1 <merge-commit>
git revert --no-commit HEAD~3..HEAD
git commit -m "Revert last 3 commits"
```

Revert adds history instead of rewriting it: the safe undo on shared branches. -m 1 keeps a merge's first-parent line.

13 Stash

```
git stash                    # save changes
git stash push -m "wip: nav" # named
git stash -u                 # + untracked
git stash list
git stash pop                # apply, drop
git stash apply stash@{1}    # apply, keep
git stash drop stash@{1}
```

Newest is stash@{0}. git stash clear deletes them all.

14 Merge conflicts

```
<<<<<<< HEAD
your changes
=======
their changes
>>>>>>> feature
```

```
git checkout --ours <file>    # keep yours
git checkout --theirs <file>  # take theirs
git add <file>                # resolved
git commit                    # finish merge
```

In a rebase, finish with git rebase --continue instead.

15 Tags

```
git tag                 # list
git tag -a v1.2.0 -m "Release 1.2.0"
git tag v1.2.0 <commit> # lightweight
git push origin v1.2.0
git push origin --tags
git tag -d v1.2.0       # delete
git push origin --delete v1.2.0
```

Use -a for releases. Tags are not pushed by default.

16 Rebase

```
git rebase main        # onto main
git rebase --continue  # after fixing
git rebase --skip      # drop this commit
git rebase --abort     # back to start
git pull --rebase origin main
```

Rebase gives a linear history but new commit SHAs. Never rebase commits already pushed to a shared branch.

17 Interactive rebase

```
git rebase -i HEAD~3       # last 3 commits
git commit --fixup=<sha>   # mark a fix
git rebase -i --autosquash main
```

pick	Keep the commit
reword	Change its message
edit	Stop to amend it
squash	Combine, keep both messages
fixup	Combine, drop this message
drop	Remove the commit

Edit the first word on each line, then save and close.

18 Cherry-pick

```
git cherry-pick <commit>
git cherry-pick <sha1> <sha2>
git cherry-pick A..B        # after A to B
git cherry-pick -x <commit> # note source
git cherry-pick -n <commit> # stage only
git cherry-pick --continue  # or --abort
```

Copies a commit with a new SHA; good for backporting fixes.