diff and patch Compare and patch files

START HERE

`diff -u <old> <new>`
Show changes as unified diff

`patch -p1 --dry-run < <name>.patch`
Test before applying

`diff -rq <dirA> <dirB>`
Which files differ?

`patch -p1 < <name>.patch`
Apply a patch

`diff -u <old> <new> > <name>.patch`
Save a patch

`patch -R -p1 < <name>.patch`
Undo a patch

`diff -ruN <old-dir> <new-dir> > <name>.patch`
Patch a whole tree

`diff --color=auto -u <old> <new>`
Colored diff

Old + new files two versions → `diff -u` → **Patch file** + and - lines → `patch -p1` → **Patched tree** new version

CONCEPTS

diff makes the patch, patch applies it. Always preview with --dry-run first.

diff exit status

Code	Meaning
0	no differences
1	files differ
2	trouble (missing file, bad option)

In a patch, a/ and b/ prefixes are path components: -p1 strips one, -p0 none.

Inside a git repository, git diff makes patches too and git apply applies them; outside one, git diff --no-index <a> <b> compares any two paths.

COMPARE FILES

`diff <a> <b>`
Show differences (normal format)

`diff -u <a> <b>`
Unified diff, the patch format

`diff -q <a> <b>`
Only say whether files differ

`diff --color=auto -u <a> <b>`
Colorize added and removed lines

`diff -y <a> <b>`
Two columns side by side

`diff -y -W 160 --suppress-common-lines <a> <b>`
Side by side, changes only, wider

`diff -U <n> <a> <b>`
Unified diff with n context lines

`diff -w <a> <b>`
Ignore all whitespace

`diff -b <a> <b>`
Ignore changes in amount of whitespace

`diff -Z <a> <b>`
Ignore trailing whitespace

`diff --strip-trailing-cr <a> <b>`
Ignore CRLF vs LF line endings

`diff -i <a> <b>`
Ignore case

`diff -B <a> <b>`
Ignore blank-line changes

`diff -I '<regex>' <a> <b>`
Ignore changes where lines match regex

COMPARE FILES (cont.)

`diff -s <a> <b>`
Also report when files are identical

`diff <(<cmd1>) <(<cmd2>)`
Compare the output of two commands
Bash/zsh process substitution; not available in plain sh.

`diff - <file>`
Compare stdin with a file

`diff -p -u <a> <b>`
Show the C function around each change

`diff --label <name> -u <a> <b>`
Use a custom name in the header

`cmp <a> <b>`
Find first differing byte (binary files)

`diff3 <mine> <base> <yours>`
Three-way comparison

COMPARE DIRECTORIES

`diff -rq <dirA> <dirB>`
List files that differ or exist on one side

`diff -r <dirA> <dirB>`
Show all differences recursively

`diff -ru <dirA> <dirB>`
Recursive unified diff

`diff -rN <dirA> <dirB>`
Show added and removed files as full diffs

`diff -rq -x '<glob>' <dirA> <dirB>`
Skip files matching a glob
Repeat -x for several patterns, e.g. -x .git -x '.o'.*

`diff -rq -X <file> <dirA> <dirB>`
Read exclude globs from a file

`diff -rq --no-dereference <dirA> <dirB>`
Compare symlinks instead of following them

`git diff --no-index <a> <b>`
Colored diff via git, no repository needed

CREATE PATCHES

`diff -u <old> <new> > <name>.patch`
Patch for a single file

CREATE PATCHES (cont.)

`diff -ruN <old-dir> <new-dir> > <name>.patch`
Patch for a directory tree

`diff -ruN -x .git <old-dir> <new-dir> > <name>.patch`
Directory patch without VCS metadata

`diff -u --label a/<file> --label b/<file> <old> <new> > <name>.patch`
Git-style names so -p1 works

`cp -r <dir> <dir>.orig`
Keep a pristine copy before editing

`diff -ruN <dir>.orig <dir> > <name>.patch`
Diff the edited tree against the copy

`git diff > <name>.patch`
Patch from a git working tree

Put the old tree first and the new one second, or the patch is reversed.

APPLY PATCHES

`patch -p1 < <name>.patch`
Apply a tree patch from its top directory

`patch <file> < <name>.patch`
Apply a patch to one named file

`patch -p1 --dry-run < <name>.patch`
Show what would happen, change nothing

`patch -R -p1 < <name>.patch`
Reverse an applied patch

`patch -p1 -i <name>.patch`
Read the patch from a file

`patch -p0 < <name>.patch`
Use the file names exactly as written

`patch -p1 -d <dir> < <name>.patch`
Apply inside another directory

`patch -N -p1 < <name>.patch`
Skip hunks that are already applied

`patch -b -p1 < <name>.patch`
Keep originals as <file>.orig

`patch -l -p1 < <name>.patch`
Tolerate whitespace differences

`patch --merge -p1 < <name>.patch`
Write conflict markers instead of .rej files

APPLY PATCHES (cont.)

`curl -sL <url> | patch -p1`
Apply a patch straight from the web

-p levels for a/src/main.c

Option	File patched
-p0	a/src/main.c
-p1	src/main.c
-p2	main.c

TROUBLESHOOTING

`find . -name '*.rej'`
Find hunks that did not apply

`cat <file>.rej`
Read a rejected hunk, then apply by hand

`patch -R --dry-run -p1 < <name>.patch`
Check whether a patch is already applied

patch applies hunks that match and rejects the rest, so a failed run can leave a half-patched tree. Use --dry-run, or a clean checkout.

Wrong -p value gives "can't find file to patch". Compare the paths in the patch header with your current directory.

"Reversed (or previously applied) patch detected" means the changes are already in; answer no, or use -N.

RECIPES

Make and apply a tree patch

```
cp -r proj proj.orig
# ... edit files in proj ...
diff -ruN proj.orig proj > fix.patch
cd /other/checkout
patch -p1 --dry-run < fix.patch && patch -p1 < fix.patch
```

Compare two directory trees

```
diff -rq -x .git -x node_modules a b
# then inspect one file
diff -u a/src/app.c b/src/app.c
```

Diff sorted output of two commands

```
diff <(sort a.txt) <(sort b.txt)
diff --color=auto -u <(ls dirA) <(ls dirB)
```

Recover from a failed patch

```
find . -name '*.rej'
# fix hunks by hand, then clean up
find . \( -name '*.orig' -o -name '*.rej' \) -delete
```