```
# Index manipulations
#--------------------
Base.@deprecate(
    permuteind(t::TensorMap,p1::IndexTuple, p2::IndexTuple=(); copy::Bool = false),
    permute(t, p1, p2; copy = copy))

"""
    permute(tsrc::AbstractTensorMap{S}, p1::NTuple{N₁,Int}, p2::NTuple{N₂,Int} =
())
        -> tdst::TensorMap{S,N₁,N₂}

Permute the indices of `tsrc::AbstractTensorMap{S}` such that a new tensor
`tdst::TensorMap{S,N₁,N₂}` is obtained, with indices in `p1` playing the role of
the
codomain or range of the map, and indices in `p2` indicating the domain.

To permute into an existing `tdst`, see [`add!`](@ref)
"""
function permute(t::TensorMap{S},
                    p1::IndexTuple, p2::IndexTuple=();
                    copy::Bool = false) where {S}
    cod = ProductSpace{S}(map(n->space(t, n), p1))
    dom = ProductSpace{S}(map(n->dual(space(t, n)), p2))

    if !copy
        # share data if possible
        if p1 === codomainind(t) && p2 === domainind(t)
            return t
        elseif has_shared_permute(t, p1, p2)
            return TensorMap(reshape(t.data, dim(cod), dim(dom)), cod, dom)
        end
    end
    # general case
    @inbounds begin
        return add!(true, t, false, similar(t, cod←dom), p1, p2)
    end
end

function permute(t::AdjointTensorMap{S}, p1::IndexTuple{N₁}, p2::IndexTuple{N₂}=();
                    copy::Bool = false) where {S,N₁,N₂}
    p1′ = map(n->adjointtensorindex(t, n), p2)
    p2′ = map(n->adjointtensorindex(t, n), p1)
    adjoint(permute(adjoint(t), p1′, p2′; copy = copy))
end

function has_shared_permute(t::TensorMap, p1, p2)
    if p1 === codomainind(t) && p2 === domainind(t)
        return true
    elseif sectortype(t) === Trivial
        stridet = i->stride(t[], i)
        sizet = i->size(t[], i)
        canfuse1, d1, s1 = TO._canfuse(sizet.(p1), stridet.(p1))
        canfuse2, d2, s2 = TO._canfuse(sizet.(p2), stridet.(p2))
        return canfuse1 && canfuse2 && s1 == 1 && (d2 == 1 || s2 == d1)
```

```
    else
        return false
    end
end

function has_shared_permute(t::AdjointTensorMap, p1, p2)
    p1′ = adjointtensorindices(t, p2)
    p2′ = adjointtensorindices(t, p1)
    return has_shared_permute(t', p1′, p2′)
end


Base.@deprecate(permuteind!(tdst::AbstractTensorMap, tsrc::AbstractTensorMap, p1,
p2),
                permute!(tdst, tsrc, p1, p2))

"""
    permute(tsrc::AbstractTensorMap{S}, p1::NTuple{N₁,Int}, p2::NTuple{N₂,Int} =
())
        -> tdst::TensorMap{S,N₁,N₂}

Permute the indices of `tsrc::AbstractTensorMap{S}` such that a new tensor
`tdst::TensorMap{S,N₁,N₂}` is obtained, with indices in `p1` playing the role of
the
codomain or range of the map, and indices in `p2` indicating the domain.

To permute into an existing `tdst`, see [`add!`](@ref)
"""
@propagate_inbounds Base.permute!(tdst::AbstractTensorMap{S,N₁,N₂},
                                  tsrc::AbstractTensorMap{S},
                                  p1::IndexTuple{N₁},
                                  p2::IndexTuple{N₂}=()) where {S,N₁,N₂} =
    add!(true, tsrc, false, tdst, p1, p2)

# Braid
function braid(t::TensorMap{S}, levels::IndexTuple,
                   p1::IndexTuple, p2::IndexTuple=();
                   copy::Bool = false) where {S}
    @assert length(levels) == numind(t)
    if BraidingStyle(sectortype(S)) isa SymmetricBraiding
        return permute(t, p1, p2; copy = copy)
    end
    if !copy && p1 == codomainind(t) && p2 == domainind(t)
        return t
    end
    # general case
    cod = ProductSpace{S}(map(n->space(t, n), p1))
    dom = ProductSpace{S}(map(n->dual(space(t, n)), p2))
    @inbounds begin
        return add!(true, t, false, similar(t, cod←dom), p1, p2, levels)
    end
end
@propagate_inbounds braid!(tdst::AbstractTensorMap{S,N₁,N₂},
                           tsrc::AbstractTensorMap{S},
```

```
                                levels::IndexTuple,
                                p1::IndexTuple{N₁},
                                p2::IndexTuple{N₂}=()) where {S,N₁,N₂} =
    add!(true, tsrc, false, tdst, p1, p2, levels)

# Transpose
function LinearAlgebra.transpose!(tdst::AbstractTensorMap, tsrc::AbstractTensorMap)
    codomain(tdst) == domain(tsrc)' && domain(tdst) == codomain(tsrc)' ||
        throw(SpaceMismatch())
    levels = (codomainind(tsrc)..., domainind(tsrc)...)
    braid!(tdst, tsrc, levels, reverse(domainind(tsrc)),
reverse(codomainind(tsrc)))
    if BraidingStyle(sectortype(tdst)) != Bosonic()
        for (c,b) in blocks(tdst)
            rmul!(b, twist(c))
        end
    end
    return tdst
end

LinearAlgebra.transpose(t::AbstractTensorMap) =
    transpose!(similar(t, domain(t)', codomain(t)'), t)

# Twist
twist(t::AbstractTensorMap, i::Int; inv::Bool = false) = twist!(copy(t), i; inv =
inv)

function twist!(t::AbstractTensorMap, i::Int; inv::Bool = false)
    N₁ = numout(t)
    for (f1,f2) in fusiontrees(t)
        θ = i <= N₁ ? twist(f1.uncoupled[i]) : twist(f2.uncoupled[i-N₁])
        rmul!(t[f1,f2], θ)
    end
    return t
end

# Fusing and splitting
```