## When To Use

Almost anything can be represented in a tree structure. Examples include directories, organization hierarchies, biological classifications, countries, etc. The `Tree` component is a way of representing the hierarchical relationship between these things. You can also expand, collapse, and select a treeNode within a `Tree`.

## Examples

### Basic

```
import React from 'react';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
        disabled: true,
        children: [
          {
            title: 'leaf',
            key: '0-0-0-0',
            disableCheckbox: true,
          },
          {
            title: 'leaf',
            key: '0-0-0-1',
          },
        ],
      },
      {
        title: 'parent 1-1',
        key: '0-0-1',
        children: [{ title: <span style={{ color: '#1677ff' }}>sss</span>,
key: '0-0-1-0' }],
      },
    ],
  },
];
```

```tsx
const App: React.FC = () => {
  const onSelect: TreeProps['onSelect'] = (selectedKeys, info) => {
    console.log('selected', selectedKeys, info);
  };

  const onCheck: TreeProps['onCheck'] = (checkedKeys, info) => {
    console.log('onCheck', checkedKeys, info);
  };

  return (
    <Tree
      checkable
      defaultExpandedKeys={['0-0-0', '0-0-1']}
      defaultSelectedKeys={['0-0-1']}
      defaultCheckedKeys={['0-0-0', '0-0-1']}
      onSelect={onSelect}
      onCheck={onCheck}
      treeData={treeData}
    />
  );
};

export default App;
```

**Controlled Tree**

```tsx
import React, { useState } from 'react';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: '0-0',
    key: '0-0',
    children: [
      {
        title: '0-0-0',
        key: '0-0-0',
        children: [
          { title: '0-0-0-0', key: '0-0-0-0' },
          { title: '0-0-0-1', key: '0-0-0-1' },
          { title: '0-0-0-2', key: '0-0-0-2' },
        ],
      },
      {
        title: '0-0-1',
```

```
        key: '0-0-1',
        children: [
          { title: '0-0-1-0', key: '0-0-1-0' },
          { title: '0-0-1-1', key: '0-0-1-1' },
          { title: '0-0-1-2', key: '0-0-1-2' },
        ],
      },
      {
        title: '0-0-2',
        key: '0-0-2',
      },
    ],
  },
  {
    title: '0-1',
    key: '0-1',
    children: [
      { title: '0-1-0-0', key: '0-1-0-0' },
      { title: '0-1-0-1', key: '0-1-0-1' },
      { title: '0-1-0-2', key: '0-1-0-2' },
    ],
  },
  {
    title: '0-2',
    key: '0-2',
  },
];

const App: React.FC = () => {
  const [expandedKeys, setExpandedKeys] = useState<React.Key[]>(['0-0-0',
'0-0-1']);
  const [checkedKeys, setCheckedKeys] = useState<React.Key[]>(['0-0-0']);
  const [selectedKeys, setSelectedKeys] = useState<React.Key[]>([]);
  const [autoExpandParent, setAutoExpandParent] = useState<boolean>(true);

  const onExpand: TreeProps['onExpand'] = (expandedKeysValue) => {
    console.log('onExpand', expandedKeysValue);
    // if not set autoExpandParent to false, if children expanded, parent
can not collapse.
    // or, you can remove all expanded children keys.
    setExpandedKeys(expandedKeysValue);
    setAutoExpandParent(false);
  };

  const onCheck: TreeProps['onCheck'] = (checkedKeysValue) => {
    console.log('onCheck', checkedKeysValue);
```

```
    setCheckedKeys(checkedKeysValue as React.Key[]);
  };

  const onSelect: TreeProps['onSelect'] = (selectedKeysValue, info) => {
    console.log('onSelect', info);
    setSelectedKeys(selectedKeysValue);
  };

  return (
    <Tree
      checkable
      onExpand={onExpand}
      expandedKeys={expandedKeys}
      autoExpandParent={autoExpandParent}
      onCheck={onCheck}
      checkedKeys={checkedKeys}
      onSelect={onSelect}
      selectedKeys={selectedKeys}
      treeData={treeData}
    />
  );
};

export default App;
```

**draggable**

```
import React, { useState } from 'react';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const x = 3;
const y = 2;
const z = 1;
const defaultData: TreeDataNode[] = [];

const generateData = (_level: number, _preKey?: React.Key, _tns?:
TreeDataNode[]) => {
  const preKey = _preKey || '0';
  const tns = _tns || defaultData;

  const children: React.Key[] = [];
  for (let i = 0; i < x; i++) {
    const key = `${preKey}-${i}`;
    tns.push({ title: key, key });
    if (i < y) {
```

```
      children.push(key);
    }
  }
  if (_level < 0) {
    return tns;
  }
  const level = _level - 1;
  children.forEach((key, index) => {
    tns[index].children = [];
    return generateData(level, key, tns[index].children);
  });
};
generateData(z);

const App: React.FC = () => {
  const [gData, setGData] = useState(defaultData);
  const [expandedKeys] = useState(['0-0', '0-0-0', '0-0-0-0']);

  const onDragEnter: TreeProps['onDragEnter'] = (info) => {
    console.log(info);
    // expandedKeys, set it when controlled is needed
    // setExpandedKeys(info.expandedKeys)
  };

  const onDrop: TreeProps['onDrop'] = (info) => {
    console.log(info);
    const dropKey = info.node.key;
    const dragKey = info.dragNode.key;
    const dropPos = info.node.pos.split('-');
    const dropPosition = info.dropPosition - Number(dropPos[dropPos.length
- 1]); // the drop position relative to the drop node, inside 0, top -1,
bottom 1

    const loop = (
      data: TreeDataNode[],
      key: React.Key,
      callback: (node: TreeDataNode, i: number, data: TreeDataNode[]) =>
void,
    ) => {
      for (let i = 0; i < data.length; i++) {
        if (data[i].key === key) {
          return callback(data[i], i, data);
        }
        if (data[i].children) {
          loop(data[i].children!, key, callback);
        }
```

```
      }
    };
    const data = [...gData];

    // Find dragObject
    let dragObj: TreeDataNode;
    loop(data, dragKey, (item, index, arr) => {
      arr.splice(index, 1);
      dragObj = item;
    });

    if (!info.dropToGap) {
      // Drop on the content
      loop(data, dropKey, (item) => {
        item.children = item.children || [];
        // where to insert. New item was inserted to the start of the array
in this example, but can be anywhere
        item.children.unshift(dragObj);
      });
    } else {
      let ar: TreeDataNode[] = [];
      let i: number;
      loop(data, dropKey, (_item, index, arr) => {
        ar = arr;
        i = index;
      });
      if (dropPosition === -1) {
        // Drop on the top of the drop node
        ar.splice(i!, 0, dragObj!);
      } else {
        // Drop on the bottom of the drop node
        ar.splice(i! + 1, 0, dragObj!);
      }
    }
    setGData(data);
  };

  return (
    <Tree
      className="draggable-tree"
      defaultExpandedKeys={expandedKeys}
      draggable
      blockNode
      onDragEnter={onDragEnter}
      onDrop={onDrop}
      treeData={gData}
```

```
    />
  );
};

export default App;
```

**load data asynchronously**

```
import React, { useState } from 'react';
import { Tree } from 'antd';

interface DataNode {
  title: string;
  key: string;
  isLeaf?: boolean;
  children?: DataNode[];
}

const initTreeData: DataNode[] = [
  { title: 'Expand to load', key: '0' },
  { title: 'Expand to load', key: '1' },
  { title: 'Tree Node', key: '2', isLeaf: true },
];

// It's just a simple demo. You can use tree map to optimize update perf.
const updateTreeData = (list: DataNode[], key: React.Key, children:
DataNode[]): DataNode[] =>
  list.map((node) => {
    if (node.key === key) {
      return {
        ...node,
        children,
      };
    }
    if (node.children) {
      return {
        ...node,
        children: updateTreeData(node.children, key, children),
      };
    }
    return node;
  });

const App: React.FC = () => {
  const [treeData, setTreeData] = useState(initTreeData);
```

```
  const onLoadData = ({ key, children }: any) =>
    new Promise<void>((resolve) => {
      if (children) {
        resolve();
        return;
      }
      setTimeout(() => {
        setTreeData((origin) =>
          updateTreeData(origin, key, [
            { title: 'Child Node', key: `${key}-0` },
            { title: 'Child Node', key: `${key}-1` },
          ]),
        );

        resolve();
      }, 1000);
    });

  return <Tree loadData={onLoadData} treeData={treeData} />;
};

export default App;
```

**Searchable**

```
import React, { useMemo, useState } from 'react';
import { Input, Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const { Search } = Input;

const x = 3;
const y = 2;
const z = 1;
const defaultData: TreeDataNode[] = [];

const generateData = (_level: number, _preKey?: React.Key, _tns?:
TreeDataNode[]) => {
  const preKey = _preKey || '0';
  const tns = _tns || defaultData;

  const children: React.Key[] = [];
  for (let i = 0; i < x; i++) {
    const key = `${preKey}-${i}`;
    tns.push({ title: key, key });
    if (i < y) {
```

```
      children.push(key);
    }
  }
  if (_level < 0) {
    return tns;
  }
  const level = _level - 1;
  children.forEach((key, index) => {
    tns[index].children = [];
    return generateData(level, key, tns[index].children);
  });
};
generateData(z);

const dataList: { key: React.Key; title: string }[] = [];
const generateList = (data: TreeDataNode[]) => {
  for (let i = 0; i < data.length; i++) {
    const node = data[i];
    const { key } = node;
    dataList.push({ key, title: key as string });
    if (node.children) {
      generateList(node.children);
    }
  }
};
generateList(defaultData);

const getParentKey = (key: React.Key, tree: TreeDataNode[]): React.Key => {
  let parentKey: React.Key;
  for (let i = 0; i < tree.length; i++) {
    const node = tree[i];
    if (node.children) {
      if (node.children.some((item) => item.key === key)) {
        parentKey = node.key;
      } else if (getParentKey(key, node.children)) {
        parentKey = getParentKey(key, node.children);
      }
    }
  }
  return parentKey!;
};

const App: React.FC = () => {
  const [expandedKeys, setExpandedKeys] = useState<React.Key[]>([]);
  const [searchValue, setSearchValue] = useState('');
  const [autoExpandParent, setAutoExpandParent] = useState(true);
```

```
  const onExpand = (newExpandedKeys: React.Key[]) => {
    setExpandedKeys(newExpandedKeys);
    setAutoExpandParent(false);
  };

  const onChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const { value } = e.target;
    const newExpandedKeys = dataList
      .map((item) => {
        if (item.title.indexOf(value) > -1) {
          return getParentKey(item.key, defaultData);
        }
        return null;
      })
      .filter((item, i, self): item is React.Key => !!(item &&
self.indexOf(item) === i));
    setExpandedKeys(newExpandedKeys);
    setSearchValue(value);
    setAutoExpandParent(true);
  };

  const treeData = useMemo(() => {
    const loop = (data: TreeDataNode[]): TreeDataNode[] =>
      data.map((item) => {
        const strTitle = item.title as string;
        const index = strTitle.indexOf(searchValue);
        const beforeStr = strTitle.substring(0, index);
        const afterStr = strTitle.slice(index + searchValue.length);
        const title =
          index > -1 ? (
            <span key={item.key}>
              {beforeStr}
              <span className="site-tree-search-value">{searchValue}</span>
              {afterStr}
            </span>
          ) : (
            <span key={item.key}>{strTitle}</span>
          );
        if (item.children) {
          return { title, key: item.key, children: loop(item.children) };
        }

        return {
          title,
          key: item.key,
```

```
        };
      });

    return loop(defaultData);
  }, [searchValue]);

  return (
    <div>
      <Search style={{ marginBottom: 8 }} placeholder="Search" onChange=
{onChange} />
      <Tree
        onExpand={onExpand}
        expandedKeys={expandedKeys}
        autoExpandParent={autoExpandParent}
        treeData={treeData}
      />
    </div>
  );
};

export default App;
```

**Tree with line**

```
import React, { useState } from 'react';
import { CarryOutOutlined, CheckOutlined, FormOutlined } from '@ant-
design/icons';
import { Select, Switch, Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    icon: <CarryOutOutlined />,
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
        icon: <CarryOutOutlined />,
        children: [
          { title: 'leaf', key: '0-0-0-0', icon: <CarryOutOutlined /> },
          {
            title: (
              <>
                <div>multiple line title</div>
```

```
              <div>multiple line title</div>
            </>
          ),
          key: '0-0-0-1',
          icon: <CarryOutOutlined />,
        },
        { title: 'leaf', key: '0-0-0-2', icon: <CarryOutOutlined /> },
      ],
    },
    {
      title: 'parent 1-1',
      key: '0-0-1',
      icon: <CarryOutOutlined />,
      children: [{ title: 'leaf', key: '0-0-1-0', icon: <CarryOutOutlined
/> }],
    },
    {
      title: 'parent 1-2',
      key: '0-0-2',
      icon: <CarryOutOutlined />,
      children: [
        { title: 'leaf', key: '0-0-2-0', icon: <CarryOutOutlined /> },
        {
          title: 'leaf',
          key: '0-0-2-1',
          icon: <CarryOutOutlined />,
          switcherIcon: <FormOutlined />,
        },
      ],
    },
  ],
},
{
  title: 'parent 2',
  key: '0-1',
  icon: <CarryOutOutlined />,
  children: [
    {
      title: 'parent 2-0',
      key: '0-1-0',
      icon: <CarryOutOutlined />,
      children: [
        { title: 'leaf', key: '0-1-0-0', icon: <CarryOutOutlined /> },
        { title: 'leaf', key: '0-1-0-1', icon: <CarryOutOutlined /> },
      ],
    },
```

```
    ],
  },
];

const App: React.FC = () => {
  const [showLine, setShowLine] = useState<boolean>(true);
  const [showIcon, setShowIcon] = useState<boolean>(false);
  const [showLeafIcon, setShowLeafIcon] = useState<React.ReactNode>(true);

  const onSelect = (selectedKeys: React.Key[], info: any) => {
    console.log('selected', selectedKeys, info);
  };

  const handleLeafIconChange = (value: 'true' | 'false' | 'custom') => {
    if (value === 'custom') {
      return setShowLeafIcon(<CheckOutlined />);
    }

    if (value === 'true') {
      return setShowLeafIcon(true);
    }

    return setShowLeafIcon(false);
  };

  return (
    <div>
      <div style={{ marginBottom: 16 }}>
        showLine: <Switch checked={!!showLine} onChange={setShowLine} />
        <br />
        <br />
        showIcon: <Switch checked={showIcon} onChange={setShowIcon} />
        <br />
        <br />
        showLeafIcon:{' '}
        <Select defaultValue="true" onChange={handleLeafIconChange}>
          <Select.Option value="true">True</Select.Option>
          <Select.Option value="false">False</Select.Option>
          <Select.Option value="custom">Custom icon</Select.Option>
        </Select>
      </div>
      <Tree
        showLine={showLine ? { showLeafIcon } : false}
        showIcon={showIcon}
        defaultExpandedKeys={['0-0-0']}
        onSelect={onSelect}
```

```
        treeData={treeData}
      />
    </div>
  );
};

export default App;
```

### Customize Icon

```
import React from 'react';
import {
  DownOutlined,
  FrownFilled,
  FrownOutlined,
  MehOutlined,
  SmileOutlined,
} from '@ant-design/icons';
import { Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    icon: <SmileOutlined />,
    children: [
      {
        title: 'leaf',
        key: '0-0-0',
        icon: <MehOutlined />,
      },
      {
        title: 'leaf',
        key: '0-0-1',
        icon: ({ selected }) => (selected ? <FrownFilled /> :
<FrownOutlined />),
      },
    ],
  },
];

const App: React.FC = () => (
  <Tree
    showIcon
    defaultExpandAll
```

```
    defaultSelectedKeys={['0-0-0']}
    switcherIcon={<DownOutlined />}
    treeData={treeData}
  />
);

export default App;
```

**directory**

```
import React from 'react';
import { Tree } from 'antd';
import type { GetProps, TreeDataNode } from 'antd';

type DirectoryTreeProps = GetProps<typeof Tree.DirectoryTree>;

const { DirectoryTree } = Tree;

const treeData: TreeDataNode[] = [
  {
    title: 'parent 0',
    key: '0-0',
    children: [
      { title: 'leaf 0-0', key: '0-0-0', isLeaf: true },
      { title: 'leaf 0-1', key: '0-0-1', isLeaf: true },
    ],
  },
  {
    title: 'parent 1',
    key: '0-1',
    children: [
      { title: 'leaf 1-0', key: '0-1-0', isLeaf: true },
      { title: 'leaf 1-1', key: '0-1-1', isLeaf: true },
    ],
  },
];

const App: React.FC = () => {
  const onSelect: DirectoryTreeProps['onSelect'] = (keys, info) => {
    console.log('Trigger Select', keys, info);
  };

  const onExpand: DirectoryTreeProps['onExpand'] = (keys, info) => {
    console.log('Trigger Expand', keys, info);
  };
```

```
  return (
    <DirectoryTree
      multiple
      draggable
      defaultExpandAll
      onSelect={onSelect}
      onExpand={onExpand}
      treeData={treeData}
    />
  );
};

export default App;
```

**Directory Debug**

Debug

```
import React from 'react';
import { Flex, Tree } from 'antd';
import type { GetProps, TreeDataNode } from 'antd';

const { DirectoryTree } = Tree;

const treeData: TreeDataNode[] = [
  {
    title: 'parent 0',
    key: '0-0',
    children: [
      { title: 'leaf 0-0', key: '0-0-0', isLeaf: true, disabled: true },
      { title: 'leaf 0-1', key: '0-0-1', isLeaf: true, disableCheckbox:
true },
    ],
  },
  {
    title: 'parent 1',
    key: '0-1',
    children: [
      { title: 'leaf 1-0', key: '0-1-0', isLeaf: true },
      { title: 'leaf 1-1', key: '0-1-1', isLeaf: true },
    ],
  },
];

const sharedProps: GetProps<typeof DirectoryTree> = {
  treeData,
```

```
  defaultExpandAll: true,
  onSelect: (keys, info) => {
    console.log('Trigger Select', keys, info);
  },
  onExpand: (keys, info) => {
    console.log('Trigger Expand', keys, info);
  },
};

const DemoOne = () => <DirectoryTree draggable defaultSelectedKeys={['0-0-
0']} />;

const DemoTwo = () => <DirectoryTree {...sharedProps} checkable
defaultSelectedKeys={['0-1-0']} />;

const DemoThree = () => (
  <DirectoryTree {...sharedProps} draggable checkable defaultSelectedKeys=
{['0-1']} />
);

const BasicDemo = () => <DirectoryTree {...sharedProps} multiple treeData=
{treeData} />;

const NormalDemo = () => <Tree {...sharedProps} defaultSelectedKeys={['0-
1']} />;

const NormalCheckDemo = () => (
  <Tree {...sharedProps} checkable defaultSelectedKeys={['0-1', '0-0-0',
'0-0-1', '0-1-1']} />
);

const NormalDragDemo = () => <Tree {...sharedProps} draggable
defaultSelectedKeys={['0-1-0']} />;

const App = () => (
  <Flex wrap gap="large">
    <DemoOne />
    <DemoTwo />
    <DemoThree />
    <BasicDemo />
    <NormalDemo />
    <NormalCheckDemo />
    <NormalDragDemo />
  </Flex>
);
```

```
export default App;
```

**Customize collapse/expand icon**

```
import React from 'react';
import { DownOutlined } from '@ant-design/icons';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
        children: [
          {
            title: 'leaf',
            key: '0-0-0-0',
          },
          {
            title: 'leaf',
            key: '0-0-0-1',
          },
          {
            title: 'leaf',
            key: '0-0-0-2',
          },
        ],
      },
      {
        title: 'parent 1-1',
        key: '0-0-1',
        children: [
          {
            title: 'leaf',
            key: '0-0-1-0',
          },
        ],
      },
      {
        title: 'parent 1-2',
        key: '0-0-2',
```

```
        children: [
          {
            title: 'leaf',
            key: '0-0-2-0',
          },
          {
            title: 'leaf',
            key: '0-0-2-1',
          },
        ],
      },
    ],
  },
];

const App: React.FC = () => {
  const onSelect: TreeProps['onSelect'] = (selectedKeys, info) => {
    console.log('selected', selectedKeys, info);
  };

  return (
    <Tree
      showLine
      switcherIcon={<DownOutlined />}
      defaultExpandedKeys={['0-0-0']}
      onSelect={onSelect}
      treeData={treeData}
    />
  );
};

export default App;
```

**Virtual scroll**

```
import React from 'react';
import { Tooltip, Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const dig = (path = '0', level = 3) => {
  const list = [];
  for (let i = 0; i < 10; i += 1) {
    const key = `${path}-${i}`;
    const treeNode: TreeDataNode = {
      title: key,
      key,
```

```
    };

    if (level > 0) {
      treeNode.children = dig(key, level - 1);
    }

    list.push(treeNode);
  }
  return list;
};

const treeData = dig();

const MemoTooltip = Tooltip || React.memo(Tooltip);

const App: React.FC = () => (
  <Tree
    treeData={treeData}
    height={233}
    defaultExpandAll
    titleRender={(item) => {
      const title = item.title as React.ReactNode;
      return <MemoTooltip title={title}>{title}</MemoTooltip>;
    }}
  />
);

export default App;
```

**Drag Debug**

Debug

```
import React from 'react';
import { CarryOutOutlined } from '@ant-design/icons';
import type { TreeDataNode, TreeProps } from 'antd';
import { Switch, Tree } from 'antd';

const x = 3;
const y = 2;
const z = 1;
const data: TreeDataNode[] = [];

const generateData = (_level: number, preKey = '0', tns = data):
TreeDataNode[] | undefined => {
  const children: string[] = [];
```

```
  for (let i = 0; i < x; i++) {
    const key = `${preKey}-${i}`;
    tns.push({ title: key, key, icon: <CarryOutOutlined /> });
    if (i < y) {
      children.push(key);
    }
  }
  if (_level < 0) {
    return tns;
  }
  const level = _level - 1;
  children.forEach((key, index) => {
    tns[index].children = [];
    return generateData(level, key, tns[index].children);
  });
};

generateData(z);

const App: React.FC = () => {
  const [gData, setGData] = React.useState<TreeDataNode[]>(data);
  const [showLine, setShowLine] = React.useState<any>(true);
  const [showIcon, setShowIcon] = React.useState<boolean>(true);
  const [showLeafIcon, setShowLeafIcon] = React.useState<boolean>(true);
  const [expandedKeys, setExpandedKeys] = React.useState<React.Key[]>(['0-
0', '0-0-0', '0-0-0-0']);

  const onDragEnter: TreeProps['onDragEnter'] = (info) => {
    console.log(info);
    // expandedKeys, set it when controlled is needed
    setExpandedKeys(info.expandedKeys);
  };

  const onDrop: TreeProps['onDrop'] = (info) => {
    console.log(info);
    const dropKey = info.node.key as number;
    const dragKey = info.dragNode.key as number;
    const dropPos = info.node.pos.split('-');
    const dropPosition = info.dropPosition - Number(dropPos[dropPos.length
- 1]);

    const loop = (
      data: TreeDataNode[],
      key: number,
      callback: (item: TreeDataNode, index: number, err: TreeDataNode[]) =>
void,
```

```
    ): void => {
      for (let i = 0; i < data.length; i++) {
        if (data[i].key === key) {
          return callback(data[i], i, data);
        }
        if (data[i].children) {
          loop(data[i].children!, key, callback);
        }
      }
    };

    const data = [...gData];

    // Find dragObject
    let dragObj: TreeDataNode;
    loop(data, dragKey, (item, index, arr) => {
      arr.splice(index, 1);
      dragObj = item;
    });

    if (!info.dropToGap) {
      // Drop on the content
      loop(data, dropKey, (item) => {
        item.children = item.children || [];
        // where to insert. New item was inserted to the end of the array
in this example, but can be anywhere
        item.children.push(dragObj);
      });
    } else if (
      ((info.node as any).props.children || []).length > 0 && // Has
children
      (info.node as any).props.expanded && // Is expanded
      dropPosition === 1 // On the bottom gap
    ) {
      loop(data, dropKey, (item) => {
        item.children = item.children || [];
        // where to insert. New item was inserted to the start of the array
in this example, but can be anywhere
        item.children.unshift(dragObj);
      });
    } else {
      let ar: TreeDataNode[];
      let i: number;
      loop(data, dropKey, (_, index, arr) => {
        ar = arr;
        i = index;
```

```
      });
      if (dropPosition === -1) {
        ar!.splice(i!, 0, dragObj!);
      } else {
        ar!.splice(i! + 1, 0, dragObj!);
      }
    }
    setGData(data);
  };

  const innerSetShowLine = (showLine: boolean) => {
    if (showLine) {
      if (showLeafIcon) {
        setShowLine({ showLeafIcon: true });
      } else {
        setShowLine(true);
      }
    } else {
      setShowLine(false);
    }
  };

  const innerSetShowLeafIcon = (showLeafIcon: boolean) => {
    setShowLeafIcon(showLeafIcon);
    setShowLine({ showLeafIcon });
  };

  return (
    <>
      <div style={{ marginBottom: 16 }}>
        showLine: <Switch checked={showLine} onChange={innerSetShowLine} />
        <br />
        <br />
        showIcon: <Switch checked={showIcon} onChange={() =>
setShowIcon(showIcon)} />
        <br />
        <br />
        showLeafIcon: <Switch checked={showLeafIcon} onChange=
{innerSetShowLeafIcon} />
      </div>
      <Tree
        showLine={showLine}
        showIcon={showIcon}
        className="draggable-tree"
        defaultExpandedKeys={expandedKeys}
        draggable
```

```
        blockNode
        onDragEnter={onDragEnter}
        onDrop={onDrop}
        treeData={gData}
      />
    </>
  );
};

export default App;
```

**Big data**

Debug

```
import React from 'react';
import { Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const treeData: TreeDataNode[] = [];

for (let i = 0; i < 100; i += 1) {
  const children: TreeDataNode[] = [];

  for (let j = 0; j < 100; j += 1) {
    children.push({
      title: `child ${i}-${j}`,
      key: `l-${i}-${j}`,
    });
  }

  treeData.push({
    title: `parent ${i}`,
    key: `l-${i}`,
    children,
  });
}

const App: React.FC = () => <Tree defaultExpandAll height={400} treeData=
{treeData} />;

export default App;
```

**Block Node**

```tsx
import React from 'react';
import { Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent',
    key: '0',
    children: [
      {
        title: 'child 1',
        key: '0-0',
        disabled: true,
      },
      {
        title: 'child 2',
        key: '0-1',
        disableCheckbox: true,
      },
    ],
  },
];

const App: React.FC = () => (
  <Tree checkable defaultSelectedKeys={['0-1']} defaultExpandAll treeData=
{treeData} blockNode />
);

export default App;
```

**Component Token**

Debug

```tsx
import React from 'react';
import { ConfigProvider, Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
```

```
        disabled: true,
        children: [
          {
            title: 'leaf',
            key: '0-0-0-0',
            disableCheckbox: true,
          },
          {
            title: 'leaf',
            key: '0-0-0-1',
          },
        ],
      },
      {
        title: 'parent 1-1',
        key: '0-0-1',
        children: [{ title: <span style={{ color: '#1677ff' }}>sss</span>,
key: '0-0-1-0' }],
      },
    ],
  },
];

const App: React.FC = () => {
  const onSelect: TreeProps['onSelect'] = (selectedKeys, info) => {
    console.log('selected', selectedKeys, info);
  };

  const onCheck: TreeProps['onCheck'] = (checkedKeys, info) => {
    console.log('onCheck', checkedKeys, info);
  };

  return (
    <ConfigProvider
      theme={{
        components: {
          Tree: {
            nodeHoverBg: '#fff2f0',
            nodeHoverColor: '#1677ff',
            nodeSelectedBg: '#ffa39e',
            nodeSelectedColor: '#fff',
            indentSize: 80,
          },
        },
      }}
    >
```

```
      <Tree
        checkable
        defaultExpandedKeys={['0-0-0', '0-0-1']}
        defaultSelectedKeys={['0-0-1']}
        defaultCheckedKeys={['0-0-0', '0-0-1']}
        onSelect={onSelect}
        onCheck={onCheck}
        treeData={treeData}
      />
    </ConfigProvider>
  );
};

export default App;
```

### Multiple lines

Debug

```
import React from 'react';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
        disabled: true,
        children: [
          {
            title: 'This is a very very very very long text',
            key: '0-0-0-0',
            disableCheckbox: true,
          },
          {
            title: 'This is also a very very very very very long text',
            key: '0-0-0-1',
          },
        ],
      },
      {
        title: 'parent 1-1',
```

```
        key: '0-0-1',
        children: [{ title: <span style={{ color: '#1677ff' }}>sss</span>,
key: '0-0-1-0' }],
      },
    ],
  },
];

const App: React.FC = () => {
  const onSelect: TreeProps['onSelect'] = (selectedKeys, info) => {
    console.log('selected', selectedKeys, info);
  };

  const onCheck: TreeProps['onCheck'] = (checkedKeys, info) => {
    console.log('onCheck', checkedKeys, info);
  };

  return (
    <Tree
      checkable
      defaultExpandedKeys={['0-0-0', '0-0-1']}
      defaultSelectedKeys={['0-0-1']}
      defaultCheckedKeys={['0-0-0', '0-0-1']}
      onSelect={onSelect}
      onCheck={onCheck}
      treeData={treeData}
      style={{ width: 200 }}
    />
  );
};

export default App;
```

## API

Common props ref：Common props

### Tree props

| Property | Description | Type | Default | Versio |
| --- | --- | --- | --- | --- |
| allowDrop | Whether to allow dropping on the node | ({ dropNode, dropPosition }) => boolean | - | |
| autoExpandParent | Whether to automatically | boolean | false | |

| | | | | |
|---|---|---|---|---|
| | expand a parent treeNode | | | |
| blockNode | Whether treeNode fill remaining horizontal space | boolean | false | |
| checkable | Add a Checkbox before the treeNodes | boolean | false | |
| checkedKeys | (Controlled) Specifies the keys of the checked treeNodes (PS: When this specifies the key of a treeNode which is also a parent treeNode, all the children treeNodes of will be checked; and vice versa, when it specifies the key of a treeNode which is a child treeNode, its parent treeNode will also be checked. When `checkable` and `checkStrictly` is true, its object has `checked` and `halfChecked` property. Regardless of whether the child or parent | string[] \| {checked: string[], halfChecked: string[]} | [] | |

| | | | | |
|---|---|---|---|---|
| | treeNode is checked, they won't impact each other | | | |
| checkStrictly | Check treeNode precisely; parent treeNode and children treeNodes are not associated | boolean | false | |
| defaultCheckedKeys | Specifies the keys of the default checked treeNodes | string[] | [] | |
| defaultExpandAll | Whether to expand all treeNodes by default | boolean | false | |
| defaultExpandedKeys | Specify the keys of the default expanded treeNodes | string[] | [] | |
| defaultExpandParent | If auto expand parent treeNodes when init | boolean | true | |
| defaultSelectedKeys | Specifies the keys of the default selected treeNodes | string[] | [] | |
| disabled | Whether disabled the tree | boolean | false | |
| draggable | Specifies whether this Tree or the node is draggable. Use `icon: false` | boolean \| ((node: DataNode) => boolean) \| { icon?: React.ReactNode \| false, nodeDraggable?: (node: DataNode) => boolean } | false | `config`: 4.17.0 |

| | | | | |
|---|---|---|---|---|
| | to disable drag handler icon | | | |
| expandedKeys | (Controlled) Specifies the keys of the expanded treeNodes | string[] | [] | |
| fieldNames | Customize node title, key, children field name | object | { title: `title`, key: `key`, children: `children` } | 4.17.0 |
| filterTreeNode | Defines a function to filter (highlight) treeNodes. When the function returns `true`, the corresponding treeNode will be highlighted | function(node) | - | |
| height | Config virtual scroll height. Will not support horizontal scroll when enable this | number | - | |
| icon | Insert a custom icon before the title. Need to set `showIcon` to true | ReactNode \| (props) => ReactNode | - | |
| loadData | Load data asynchronously | function(node) | - | |
| loadedKeys | (Controlled) Set loaded tree nodes. Need work with `loadData` | string[] | [] | |

| | | | | |
|---|---|---|---|---|
| multiple | Allows selecting multiple treeNodes | boolean | false | |
| rootStyle | Style on the root element | CSSProperties | - | 4.20.0 |
| selectable | Whether can be selected | boolean | true | |
| selectedKeys | (Controlled) Specifies the keys of the selected treeNodes, multiple selection needs to set `multiple` to true | string[] | - | |
| showIcon | Controls whether to display the `icon` node, no default style | boolean | false | |
| showLine | Shows a connecting line | boolean \| {showLeafIcon: ReactNode \| ((props: AntTreeNodeProps) => ReactNode)} | false | |
| switcherIcon | Customize expand/collapse icons for tree nodes (With default rotate angular style) | ReactNode \| ((props: AntTreeNodeProps) => ReactNode) | - | renderPr 4.20.0 |
| switcherLoadingIcon | Customize loading icons for tree nodes | ReactNode | - | 5.20.0 |
| titleRender | Customize tree node title render | (nodeData) => ReactNode | - | 4.5.0 |
| treeData | The treeNodes data Array, if | array<{ key, title, children, [disabled, | - | |

| | | | | |
|---|---|---|---|---|
| | set it then you need not to construct children TreeNode. (key should be unique across the whole array) | selectable] }> | | |
| virtual | Disable virtual scroll when set to false | boolean | true | 4.1.0 |
| onCheck | Callback function for when the onCheck event occurs | function(checkedKeys, e:{checked: boolean, checkedNodes, node, event, halfCheckedKeys}) | - | |
| onDragEnd | Callback function for when the onDragEnd event occurs | function({event, node}) | - | |
| onDragEnter | Callback function for when the onDragEnter event occurs | function({event, node, expandedKeys}) | - | |
| onDragLeave | Callback function for when the onDragLeave event occurs | function({event, node}) | - | |
| onDragOver | Callback function for when the onDragOver event occurs | function({event, node}) | - | |
| onDragStart | Callback function for when the onDragStart event occurs | function({event, node}) | - | |

| | | | | |
|---|---|---|---|---|
| onDrop | Callback function for when the onDrop event occurs | function({event, node, dragNode, dragNodesKeys}) | - | |
| onExpand | Callback function for when a treeNode is expanded or collapsed | function(expandedKeys, {expanded: boolean, node}) | - | |
| onLoad | Callback function for when a treeNode is loaded | function(loadedKeys, {event, node}) | - | |
| onRightClick | Callback function for when the user right clicks a treeNode | function({event, node}) | - | |
| onSelect | Callback function for when the user clicks a treeNode | function(selectedKeys, e:{selected: boolean, selectedNodes, node, event}) | - | |

**TreeNode props**

| Property | Description | Type | Default | |
|---|---|---|---|---|
| checkable | When Tree is checkable, set TreeNode display Checkbox or not | boolean | - | |
| disableCheckbox | Disables the checkbox of the treeNode | boolean | false | |
| disabled | Disables the treeNode | boolean | false | |
| icon | Customize icon. When you pass component, whose render will receive full TreeNode props as component props | ReactNode \| (props) => ReactNode | - | |

| | | | | |
|---|---|---|---|---|
| isLeaf | Determines if this is a leaf node(effective when `loadData` is specified). `false` will force trade TreeNode as a parent node | boolean | - | |
| key | Used with (default)ExpandedKeys / (default)CheckedKeys / (default)SelectedKeys. P.S.: It must be unique in all of treeNodes of the tree | string | (internal calculated position of treeNode) | |
| selectable | Set whether the treeNode can be selected | boolean | true | |
| title | Title | ReactNode | --- | |

**DirectoryTree props**

| Property | Description | Type | Default |
|---|---|---|---|
| expandAction | Directory open logic, optional: false \| `click` \| `doubleClick` | string \| boolean | `click` |

## Note

Before `3.4.0` : The number of treeNodes can be very large, but when `checkable=true` , it will increase the compute time. So, we cache some calculations (e.g. `this.treeNodesStates` ) to avoid double computing. But, this brings some restrictions. **When you load treeNodes asynchronously, you should render tree like this**:

```
{
  this.state.treeData.length ? (
    <Tree>
      {this.state.treeData.map((data) => (
        <TreeNode />
      ))}
    </Tree>
  ) : (
    'loading tree'
  );
}
```

**Tree Methods**

| Name | Description |
|---|---|

| | |
|---|---|
| scrollTo({ key: string \| number; align?: 'top' \| 'bottom' \| 'auto'; offset?: number }) | Scroll to key item in virtual scroll |

## Design Token

## FAQ

### Why defaultExpandAll not working on ajax data?

`default` prefix prop only works when initializing. So `defaultExpandAll` has already executed when ajax load data. You can control `expandedKeys` or render Tree when data loaded to realize expanded all.

### Virtual scroll limitation

Virtual scroll only render items in visible region. Thus not support auto width (like long `title` with horizontal scroll).

### What does `disabled` node work logic in the tree?

Tree change its data by conduction. Includes checked or auto expanded, it will conduction state to parent / children node until current node is `disabled`. So if a controlled node is `disabled`, it will only modify self state and not affect other nodes. For example, a parent node contains 3 child nodes and one of them is `disabled`. When check the parent node, it will only check rest 2 child nodes. As the same, when check these 2 child node, parent will be checked whatever checked state the `disabled` one is.

This conduction logic prevent that modify `disabled` parent checked state by check children node and user can not modify directly with click parent which makes the interactive conflict. If you want to modify this conduction logic, you can customize it with `checkStrictly` prop.