??????????

- lib.mapAttrsToList
- lib.mkMerge
- lib.mkIf
- lib.genAttrs

lib.mapAttrsToList

??????????, ???????? ? ????????????

`lib.mapAttrsToList` — это функция, которая **преобразует набор атрибутов в список, применяя функцию к каждой паре ключ-значение.**

```
lib.mapAttrsToList f attrs
```

Описание:

- `f` — функция, принимающая два аргумента: `name` (ключ) и `value` (значение);
- `attrs` — набор атрибутов.

Возвращает **список** результатов применения `f` к каждому элементу.

Использование:

- Генерация конфигурационных файлов.
- Создания списков для динамической конфигурации.
- Преобразования данных между форматами.

???????

????????????? ?????? ? ?????? ?????

Исходный код:

```
{ lib }:
let
  users = {
    alice = 25;
    bob = 30;
    charlie = 35;
  };

  result = lib.mapAttrsToList (name: age: "${name} is ${toString age} years old") users;
in
result
```

Результат:

```
[ "alice is 25 years old" "bob is 30 years old" "charlie is 35 years old" ]
```

???????? ?????? ?????????????

```
{ lib, ... }:
let
  userConfigs = {
    alice = {
      uid = 1001;
      group = "users";
      shell = "/bin/bash";
    };
    bob = {
      uid = 1002;
      group = "developers";
      shell = "/bin/zsh";
    };
  };

  # Преобразуем в формат, понятный для users.users
  usersList = lib.mapAttrsToList (username: config: {
    name = username;
    inherit (config) uid group shell;
    isNormalUser = true;
  }) userConfigs;
in
{
  users.users = builtins.listToAttrs (map (u: { name = u.name; value = u; }) usersList);
}
```

???????? ???????????????? ?????? ?? ??????

```
{ lib, ... }:
let
  services = {
    nginx = {
      port = 80;
      enable = true;
```

```
    };
    postgres = {
      port = 5432;
      enable = true;
    };
    redis = {
      port = 6379;
      enable = false;
    };
  };

  # Фильтруем и создаем конфиги только для включенных сервисов
  enabledServices = lib.mapAttrsToList (name: config:
    lib.optional config.enable {
      serviceName = name;
      inherit (config) port;
      configFile = ./${name}-config.conf;
    }
  ) services;
in
{
  # enabledServices будет списком списков, flatten его
  services = lib.flatten enabledServices;
}
```

????????? firewall

```
{ lib, ... }:
let
  openPorts = {
    http = 80;
    https = 443;
    ssh = 22;
    smtp = 25;
  };

  firewallRules = lib.mapAttrsToList (name: port: {
    # Создаем правило для каждого порта
    rule = "-p tcp --dport ${toString port} -j ACCEPT";
    description = "Allow ${name} (port ${toString port})";
```

```
  }) openPorts;
in
{
  networking.firewall.extraCommands = lib.concatMapStringsSep "\n" (rule:
    "iptables -A INPUT ${rule.rule} # ${rule.description}"
  ) firewallRules;
}
```

????????? ? ???????? ?????????

```
{ lib }:
let
  data = { a = 1; b = 2; c = 3; };
in
{
  # mapAttrsToList: возвращает список
  mapAttrsToList = lib.mapAttrsToList (n: v: "${n}=${toString v}") data;
  # Результат: [ "a=1" "b=2" "c=3" ]

  # mapAttrs: возвращает набор
  mapAttrs = lib.mapAttrs (n: v: v * 2) data;
  # Результат: { a = 2; b = 4; c = 6; }

  # attrValues: только значения
  attrValues = lib.attrValues data;
  # Результат: [ 1 2 3 ]
}
```

????????? systemd ?????

```
{ lib, ... }:
let
  backupJobs = {
    "backup-home" = {
      source = "/home";
      destination = "/backup/home";
      schedule = "daily";
    };
    "backup-etc" = {
      source = "/etc";
```

```
      destination = "/backup/etc";
      schedule = "weekly";
    };
  };

  systemdServices = lib.mapAttrsToList (jobName: config:
    {
      name = "backup-${jobName}";
      value = {
        description = "Backup ${config.source}";
        script = ''
          rsync -av ${config.source} ${config.destination}
        '';
        startAt = config.schedule;
      };
    }
  ) backupJobs;
in
{
  systemd.services = builtins.listToAttrs systemdServices;
}
```

lib.mkMerge

??????????, ???????? ? ????????????

`lib.mkMerge` — это функция, которая позволяет **объединять несколько наборов атрибутов в один**, решая конфликты через **приоритеты**.
Это особенно полезно в модульной конфигурации NixOS, когда вы хотите разделить конфигурацию на части и объединить их без ручного разрешения конфликтов.
Когда вы объединяете наборы атрибутов в Nix, могут возникать конфликты (одинаковые ключи с разными значениями). `lib.mkMerge` решает эти конфликты, используя систему приоритетов: каждый элемент получает приоритет (по умолчанию 0), и выигрывает значение с наивысшим приоритетом.

???????

????????

Исходный код

```
{ lib, ... }:

let
  # Три конфигурации, которые мы хотим объединить
  config1 = { boot.loader.grub.enable = true; };
  config2 = { boot.loader.systemd-boot.enable = true; };
  config3 = { networking.hostName = "myhost"; };
in
lib.mkMerge [
  config1
  config2  # Конфликт с config1 по boot.loader.*
  config3
]
```

Описание

В этом примере `config1` и `config2` конфликтуют (оба определяют загрузчик). По умолчанию `lib.mkMerge` просто выберет последнее значение, но с приоритетами можно контролировать результат.

?????????? ??????????

```
{ lib, ... }:

lib.mkMerge [
  # Приоритет 1000 (высокий)
  (lib.mkIf false {
    services.nginx.enable = true;
  })

  # Приоритет 100 (средний)
  {
    services.nginx.enable = lib.mkDefault false;
    services.nginx.virtualHosts."example.com".root = "/var/www";
  }

  # Приоритет 150 (выше среднего)
  (lib.mkForce {
    services.nginx.enable = true;  # Переопределит mkDefault
  })
]
```

Описание:

- `mkIf` имеет приоритет 1000
- `mkDefault` имеет приоритет 100
- `mkForce` имеет приоритет 150
- Результат: `services.nginx.enable = true` (побеждает `mkForce`)

????????? ????

```
{ lib, config, ... }:

let
  commonNetwork = {
    networking.networkmanager.enable = true;
    networking.firewall.enable = true;
  };

  homeConfig = {
    networking.hostName = "home-pc";
    networking.firewall.allowedTCPPorts = [ 80 443 ];
  };
```

```
  workConfig = {
    networking.hostName = "work-laptop";
    networking.firewall.allowedTCPPorts = [ 22 3389 ];
    networking.proxy.default = "http://proxy.company.com:8080";
  };

  # Динамически выбираем конфигурацию
  environmentConfig = if config.isWorkEnvironment then workConfig else homeConfig;
in
{
  imports = [ ./hardware-configuration.nix ];

  options.isWorkEnvironment = lib.mkOption {
    type = lib.types.bool;
    default = false;
  };

  config = lib.mkMerge [
    commonNetwork
    environmentConfig
    {
      # Этот блок имеет самый высокий приоритет
      networking.nameservers = lib.mkForce [ "1.1.1.1" "8.8.8.8" ];
    }
  ];
}
```

?????? ??? ?????? ?????????? ????????????

```
# hardware/base.nix
{ lib, ... }:

{
  options.hardware = {
    profile = lib.mkOption {
      type = lib.types.enum [ "desktop" "laptop" "server" ];
      default = "desktop";
    };
```

```
    hasBluetooth = lib.mkOption {
      type = lib.types.bool;
      default = false;
    };
  };

  config = lib.mkMerge [
    # Базовая конфигурация для всех систем
    {
      hardware.enableRedistributableFirmware = true;
      powerManagement.enable = true;
    }

    # Конфигурация для ноутбуков
    (lib.mkIf (config.hardware.profile == "laptop") {
      services.tlp.enable = true;
      services.auto-cpufreq.enable = true;
      hardware.hasBluetooth = lib.mkDefault true;
    })

    # Конфигурация для серверов
    (lib.mkIf (config.hardware.profile == "server") {
      powerManagement.enable = lib.mkForce false;  # Отключаем на серверах
      services.openssh.enable = true;
    })

    # Конфигурация Bluetooth
    (lib.mkIf config.hardware.hasBluetooth {
      hardware.bluetooth.enable = true;
      services.blueman.enable = true;
    })
  ];
}
```

????????? ???????????

```
{ lib, ... }:

lib.mkMerge [
```

```
  {
    services.postgresql = lib.mkMerge [
      {
        enable = true;
        package = pkgs.postgresql_15;
      }
      (lib.mkIf config.services.grafana.enable {
        authentication = ''
          host grafana all ::1/128 md5
        '';
      })
    ];
  }

  {
    environment.systemPackages = with pkgs; [
      vim
      htop
    ];
  }
]
```

?????? ???????????

Приоритеты по умолчанию (от высокого к низкому):

1. `lib.mkIf` (1000) — условное включение
2. `lib.mkOverride` — явное указание приоритета
3. `lib.mkForce` (50) — принудительное значение
4. `lib.mkDefault` (1000 для false, 100 для true) — значения по умолчанию
5. Обычные значения (0) — стандартный приоритет

?????? ???????

1. **Порядок важен только при равных приоритетах** — при одинаковых приоритетах побеждает последнее значение
2. **Глубокое слияние** — `mkMerge` рекурсивно объединяет вложенные атрибуты
3. **Не путать с `mkOverride`** — `mkMerge` объединяет списки наборов, а `mkOverride` изменяет приоритет конкретного атрибута

??????? ?????????

```
# Объединение конфигураций
config = lib.mkMerge [
  commonConfig
  (lib.mkIf condition conditionalConfig)
  (lib.mkForce forcedConfig)
  userConfig
];

# Эквивалентно (но более читаемо и модульно):
# {
#   commonConfig
#   conditionalConfig (если condition == true)
#   forcedConfig (с приоритетом)
#   userConfig
# }
```

lib.mkIf

???????????, ???????? ? ????????????

`lib.mkIf` — это функция, используемая для **условного определения** конфигурационных опций в модулях.

```
lib.mapAttrsToList condition definition
```

Описание:

- `condition` — логическое выражение (true/false);
- `deffinition` — значение или набор атрибутов, которые будут применены, если условие истинно.

Возвращает `definition`, если `condition = true`, иначе возвращает особое значение "пустоты".

Использование:

- Позволяет включать и выключать части конфигурации на основе условий.

???????

??????? ???????

Исходный код:

```
{ config, lib, ... }:
{
  config = lib.mkIf (config.networking.hostName == "webserver") {
    services.nginx.enable = true;
    services.nginx.virtualHosts."example.com".root = "/var/www";
  };
}
```

????????? ???????

```
{ config, lib, ... }:
{
```

```
  config = lib.mkIf config.services.xserver.enable {
    sound.enable = true;

    hardware.pulseaudio = lib.mkIf config.services.pipewire.enable {
      enable = false;
    };
  };
}
```

?????????????? ? `mkMerge`

```
{ config, lib, ... }:
{
  config = lib.mkMerge [
    # Общие настройки
    {
      environment.systemPackages = with pkgs; [ vim wget ];
    }

    # Условные настройки для сервера
    (lib.mkIf (config.networking.hostName == "server") {
      services.openssh.enable = true;
      services.nginx.enable = true;
    })

    # Условные настройки для рабочей станции
    (lib.mkIf config.services.xserver.enable {
      services.xserver.desktopManager.gnome.enable = true;
      hardware.bluetooth.enable = true;
    })
  ];
}
```

???????? ????????? ??????

```
{ config, lib, ... }:
{
  imports = [
    (lib.mkIf config.virtualisation.docker.enable ./docker-extra.nix)
    (lib.mkIf config.services.mysql.enable ./mysql-backup.nix)
```

```
  ];
}
```

??????? ???????

```
{ config, lib, ... }:
let
  isProduction = config.networking.hostName == "prod-server";
  hasGPU = config.hardware.opengl.enable;
in
{
  config = lib.mkIf (isProduction && hasGPU) {
    services.tensorflow-serving.enable = true;
    services.cuda.enable = true;
  };
}
```

???????? ???????? ? ??????

```
{ config, lib, ... }:
{
  options.myService = {
    enable = lib.mkEnableOption "My custom service";
    extraConfig = lib.mkOption {
      type = lib.types.str;
      default = "";
    };
  };

  config = lib.mkIf config.myService.enable {
    systemd.services.my-service = {
      description = "My Service";
      wantedBy = [ "multi-user.target" ];
      script = ''
        echo "Starting my service"
        ${lib.optionalString (config.myService.extraConfig != "") ''
          echo "Extra config: ${config.myService.extraConfig}"
        ''}
      '';
    };
```

```
  };
}
```

??????????? ????????????

?????????? ???????????

```
# Правильно:
config = lib.mkIf condition1 {
  services.xyz = lib.mkIf condition2 {
    enable = true;
  };
};

# Неправильно (может вызвать ошибки):
config = {
  services.xyz = lib.mkIf condition1 {
    enable = lib.mkIf condition2 true;
  };
};
```

?????? ? `mkOverride`

```
{ config, lib, ... }:
{
  config = lib.mkIf config.services.foo.enable {
    services.foo.configFile = lib.mkOverride 90 "/etc/foo/custom.conf";
    # Приоритет 90 (меньше = выше приоритет)
  };
}
```

????????????? ? `mkDefault` ? `mkForce`

```
{ config, lib, ... }:
{
  config = lib.mkIf config.networking.wireless.enable {
    networking.networkmanager.wifi.backend = lib.mkDefault "iwd";
    # Установит значение только если оно не было задано явно
  };
}
```

???????? ???????????? ???????

```
{ config, lib, pkgs, ... }:
let
  roles = {
    web = config.node.role.web or false;
    db = config.node.role.db or false;
    cache = config.node.role.cache or false;
  };
in
{
  options.node.role = {
    web = lib.mkEnableOption "Web server role";
    db = lib.mkEnableOption "Database server role";
    cache = lib.mkEnableOption "Cache server role";
  };

  config = lib.mkMerge [
    # Базовая конфигурация для всех узлов
    {
      environment.systemPackages = with pkgs; [ htop tmux ];
      services.openssh.enable = true;
    }

    # Конфигурация для веб-сервера
    (lib.mkIf roles.web {
      services.nginx.enable = true;
      services.phpfpm.enable = true;
      networking.firewall.allowedTCPPorts = [ 80 443 ];
    })

    # Конфигурация для БД сервера
    (lib.mkIf roles.db {
      services.postgresql.enable = true;
      services.postgresql.ensureDatabases = [ "appdb" ];
      networking.firewall.allowedTCPPorts = [ 5432 ];
    })

    # Конфигурация для кэш-сервера
    (lib.mkIf roles.cache {
```

```
      services.redis.enable = true;
      networking.firewall.allowedTCPPorts = [ 6379 ];
    })
  ];
}
```

?????? ?????????

1. **Ленивые вычисления**: `lib.mkIf` использует ленивые вычисления, поэтому определение вычисляется только если условие истинно.
2. **Обработка ошибок**: Условие должно быть полностью определено. Нельзя использовать значения, которые могут быть `undefined`.
3. **Композиция**: `mkIf` хорошо сочетается с другими функциями библиотеки: `mkMerge`, `mkOption`, `mkDefault`.
4. **Читаемость**: Для сложных условий рекомендуется использовать `let`-блоки для присвоения имен условиям.
5. **Отладка**: Если нужно увидеть, какие условия срабатывают, можно использовать `lib.traceIf` для отладки.

??????? ???????

```
{ config, lib, ... }:
{
  config = lib.mkIf (lib.traceValFn (v: "Condition value: ${toString v}")
    (config.services.nginx.enable)) {
    # конфигурация
  };
}
```

lib.genAttrs

`lib.genAttrs` — функция, которая принимает список имён и функцию-генератор, создавая набор атрибутов, где каждый элемент списка становится ключом, а значение вычисляется функцией-генератором на основе этого ключа.
Инструмент для избежания повторения в конфигурациях NixOS, особенно когда нужно применить похожую конфигурацию к множеству сущностей(службы, пользователи, способы взаимодействия(interfaces))

?????????

```
genAttrs :: [String] -> (String -> Any) -> AttrSet
```

- первый аргумент — список строк (имена будущих атрибутов);
- второй аргумент — функция, принимающая имя и возвращающая значение для этого атрибута;

Возвращает набор атрибутов `{ name1 = value1; name2 = value2; ... }`

???????

????????

```
{ lib }:
let
  names = [ "foo" "bar" "baz" ];
  # Функция-генератор: добавляет "-suffix" к имени
  addSuffix = name: "${name}-suffix";

  result = lib.genAttrs names addSuffix;
in
result
```

Результат:

```
{
  foo = "foo-suffix";
  bar = "bar-suffix";
  baz = "baz-suffix";
}
```

??????? ????????? ?????

```
{ config, lib, pkgs, ... }:
let
  myServices = [ "nginx" "postgresql" "redis" ];

  # Создаём атрибутный набор включённых сервисов
  enabledServices = lib.genAttrs myServices (name: {
    enable = true;
  });
in
{
  # Включаем все сервисы одним выражением
  services = enabledServices;

  # Эквивалентно:
  # services.nginx.enable = true;
  # services.postgresql.enable = true;
  # services.redis.enable = true;
}
```

??????? ????????????

```
{ config, lib, ... }:
let
  users = [ "alice" "bob" "charlie" ];

  # Создаём базовую конфигурацию для каждого пользователя
  userConfigs = lib.genAttrs users (name: {
    isNormalUser = true;
    extraGroups = [ "wheel" "networkmanager" ];
    createHome = true;
    home = "/home/${name}";
  });
in
{
  users.users = userConfigs;
}
```

????????? ???????

```
{ pkgs, lib, ... }:
let
  languages = [ "python3" "go" "nodejs" "rustc" ];

  # Устанавливаем последние версии языков программирования
  devPackages = lib.genAttrs languages (name: pkgs.${name});
in
{
  environment.systemPackages = builtins.attrValues devPackages;

  # Или с дополнительной конфигурацией:
  languages = lib.genAttrs languages (name: {
    enable = true;
    package = pkgs.${name};
  });
}
```

????????? ???????????????? ??????

```
{ config, lib, ... }:
let
  domains = [ "example.com" "test.com" "admin.example.com" ];

  nginxVhosts = lib.genAttrs domains (domain: {
    serverName = domain;
    root = "/var/www/${domain}";
    locations."/".proxyPass = "http://localhost:8080";
    enableSSL = true;
    sslCertificate = "/var/ssl/${domain}.crt";
    sslCertificateKey = "/var/ssl/${domain}.key";
  });
in
{
  services.nginx.virtualHosts = nginxVhosts;
}
```

?????????? ?????? ? ???????????? ?? ?????

```
{ lib, ... }:
let
```

```
  interfaces = [ "eth0" "eth1" "wlan0" ];

  networkConfig = lib.genAttrs interfaces (name:
    if lib.hasPrefix "eth" name then {
      useDHCP = false;
      ipv4.addresses = [{
        address = "192.168.1.${toString (10 + lib.elemIndex name interfaces)}";
        prefixLength = 24;
      }];
    } else {
      useDHCP = true;
      wireless.enable = true;
    }
  );
in
{
  networking.interfaces = networkConfig;
}
```

??????? ? `listToAttrs`

Часто путают `genAttrs` с `listToAttrs`. Вот ключевое отличие:

```
# genAttrs - проще, когда нужно создать значения на основе имён
lib.genAttrs [ "a" "b" ] (name: "value-${name}")
# => { a = "value-a"; b = "value-b"; }

# listToAttrs - когда у вас уже есть пары {name, value}
lib.listToAttrs [
  { name = "a"; value = 1; }
  { name = "b"; value = 2; }
]
# => { a = 1; b = 2; }
```

?????????????

????????????? ? ??????? ?????????

```
lib.genAttrs (lib.filter (s: lib.hasPrefix "dev" s) allNames) (name: ...)
```

????????????? ? ??????? NixOS

```
options = lib.genAttrs [ "foo" "bar" ] (name: lib.mkOption {
  type = lib.types.str;
  default = name;
});
```

???????????? ???????? ????????? ?? ??????

```
let
  keys = builtins.attrNames someInputSet;
  processed = lib.genAttrs keys (key: transformFunction someInputSet.${key});
in
processed
```