**Multiple Github Accounts on One Computer with SSH**

<https://www.youtube.com/watch?v=R09UNc4ZNi4>

<https://www.youtube.com/watch?v=pE3EuiyShoM>

<https://www.youtube.com/watch?v=ap56ivm0dhw>

!!! Plan is => to implement in file config other ssh\_key , for example for AWS, Azure accounts.

<https://linux.die.net/man/1/ssh>

**-o** *option*  
Can be used to give options in the format used in the configuration file (**~/.ssh/config**). This is useful for specifying options for which there is no separate command-line flag. For full details of the options listed below, and their possible values, see **[ssh\_config](https://linux.die.net/man/5/ssh_config)**(5).

Managing multiple Git identities is common for individuals who work on various projects or with different clients. Here are examples of how you can set up and manage multiple Git identities:

**1. Configure Global Settings:**

Set your global Git configuration with your default personal details:

git config --global user.name "Your Name"

git config --global user.email [your.email@example.com](mailto:your.email@example.com)

**2. Create SSH Keys:**

Windows has preinstalled SSH (OpenSSH), check if active on your computer in CMD-terminal:

ssh -v

Generate SSH keys for each identity/Application/Project changing parameter: -C “edit\_as\_needed” as a label of usage:

ssh-keygen -t ed25519 -C "ukriggo\_at\_KS\_laptop"

Save each key in the directory **~/.ssh/ (**c:\Users\YOU\.ssh\**)** with a different name, like “**id\_personal”**, “**id\_work”**, etc.

**3. Configure SSH Config:**

Create a new file **config** => Use **~/.ssh/config** to manage different identities:

# Personal Identity

Host personal.github

HostName github.com

User git

IdentityFile ~/.ssh/id\_rsa\_personal

# Work Identity

Host work\_github

HostName github.com

User git

IdentityFile ~/.ssh/id\_rsa\_work

!!! ===> TEMPLATE for different SSH connections – param meaning link **[ssh\_config](https://linux.die.net/man/5/ssh_config)**:

Host **ІМЯ** , де ІМЯ це псевдонім (alias) для цього ssh-підкючення

HostName github.com – сайт, до якого буде ssh підкючення (use IP or FQDN )

User git - user for SSH connection, in this case , user is “git” (for github & gitlab)

Port 2022 - if using port 2022 instead of port 22 for ssh connection

IdentityFile ~/.ssh/КЛЮЧ - публічний ключ для ssh підкючення ()

IdentitiesOnly yes – will be used the authentication identity files configured in the ssh\_config

**IdentityFile** - The file name may use the tilde syntax to refer to a user's home directory or one of the following escape characters: '%d' (local user's home directory), '%u' (local user name), '%l' (local host name), '%h' (remote host name) or '%r' (remote user name).

It is possible to have multiple identity files specified in configuration files; all these identities will be tried in sequence.

**=-=-=-Igor Gor 10/01/2014: ~/.ssh/config** **=-=-=**

# ukriggo@ukr.net => Personal private ssh identity for github.com

# git remote add origin git@ghub:ukrIgG/SSH-management-study-.git

# using alias ghub for ssh connetion

**Host ghub**

**HostName github.com**

**User git**

**IdentityFile** **~/.ssh/IgGo\_github**

**IdentitiesOnly yes**

# ukriggo@ukr.net => Personal private ssh identity for gitlab.com

# git remote add origin git@glab:ukriggo/gitlab\_SSH\_connect.git

# using alias glab for ssh connetion

**Host glab**

**HostName gitlab.com**

**User git**

# Port - if using not default port 22, but other

**IdentityFile ~/.ssh/IgGo\_gitlab**

**IdentitiesOnly yes**

**==-=-=-=-========-=-=-=-=-==-=-=-=-=**

**Далі можна відправити публічний ключ на віддалений (remote) сервер для SSH-passwordless\_authentication.** this OpenSSH script used to install your public key on a remote machine, allowing you to log-in without a password. Перший метод (треба спобувати!!!)

* Після створення віддаленого сервера зайти в свою ~/.ssh directory та створити файл files inside «**authorized\_keys**», та потім вставити у цей файл **authorized\_keys** контент other\_system.pub (It is used to specify the public keys that are allowed to connect to a user account on a remote machine.)
* After following these steps, you should be able to SSH to the remote system without being prompted for a password.

**ssh user@remote\_system**

The second method to make passwordless SSH to remote server:

ssh-copy-id -I ~/.ssh/other\_system.pub xx-remote.server.com

# if using other (port 2022) then port 22 for ssh connection

ssh-copy-id -I ~/.ssh/other\_system.pub -p 2022 xx-remote.server.com

* **-i**: Specifies the identity file (your public key).
* **~/.ssh/other\_system.pub**: The path to your public key file.
* **user**: Your username on the remote system
* **xx-remote.server.com**: The IP address or hostname of the remote system.
* **Permissions:** Ensure that the permissions on your **~/.ssh** directory are set correctly. The directory should have permission **700**, and the files inside (such as **authorized\_keys**) should have permission **600**.

chmod 700 ~/.ssh

chmod 600 ~/.ssh/authorized\_keys

* **SSH Agent:** If you are still prompted for a password, make sure your SSH agent is running and that your private key is added to the agent:

eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id\_rsa

After following these steps, you should be able to SSH to the remote system without being prompted for a password.

ssh user@remote\_system

**4. Working with Repositories:**

When cloning, **use the specific hostname** defined in your SSH **config**:

$ git clone git@**glab**:ukriggo/gitlab\_SSH\_connect.git = using **glab** alias

Or simply PUSH commit from local to remote (github.com or gitlab.com) using:

$ git push -u origin main

**5. Local Repository Configuration:**

For individual repositories, set local configuration:

# Inside the repository

git config user.name "Your Name"

git config user.email "your.email@example.com"

**6. Use Git Worktrees:**

If working on multiple branches simultaneously, leverage Git worktrees:

# Create a worktree for each branch

git worktree add -b feature-branch1 ../path/to/branch1

git worktree add -b feature-branch2 ../path/to/branch2

**7. Use Git Credentials Manager:**

Utilize a credential manager like **git-credential-manager** to handle credentials:

git-credential-manager install

**8. Switching Identities:**

To switch between identities, update your global Git configuration:

# Set global identity for personal work

git config --global user.name "Your Personal Name"

git config --global user.email "your.personal@email.com"

# Set global identity for work

git config --global user.name "Your Work Name"

git config --global user.email "your.work@email.com"

**9. Aliases for Cloning:**

Create aliases in your shell for easy cloning:

alias clone-personal='git clone git@personal.github.com:user/repo.git'

alias clone-work='git clone git@work.github.com:company/repo.git'

Now, you can easily manage and switch between different Git identities based on your projects, clients, or personal/work-related activities. Adjust these examples according to your specific needs and preferences.

Below are examples of **.gitconfig** file with configurations for managing multiple Git identities. You can customize these configurations based on your specific details:

# Global Configuration

[user]

name = Your Global Name

email = [your.global@email.com](mailto:your.global@email.com)

[core]

autocrlf = input

# Personal Identity

[includeIf "gitdir:~/path/to/personal/"]

path = ~/path/to/personal/.gitconfig

# Work Identity

[includeIf "gitdir:~/path/to/work/"]

path = ~/path/to/work/.gitconfig

**Personal Identity Configuration (~/.gitconfig\_personal):**

[user]

name = Your Personal Name

email = [your.personal@email.com](mailto:your.personal@email.com)

[core]

autocrlf = input

**Work Identity Configuration (~/.gitconfig\_work):**

[user]

name = Your Work Name

email = [your.work@email.com](mailto:your.work@email.com)

[core]

autocrlf = input

Adjust the paths and details based on your actual directory structure. This setup uses the **includeIf** directive to include different configurations based on the directory where the Git repository resides. This way, you can have specific configurations for personal and work projects.

=-=-=-=-=-=-=-

Here's example ~/.gitconfig file:

[user]

name = Garrit Franke

email = garrit@slashdev.space

[includeIf "gitdir:~/work/"]

path = ~/.gitconfig-work

[includeIf "gitdir:~/work/client2/"]

path = ~/.gitconfig-client2

[includeIf "gitdir:~/sources/"]

path = ~/.gitconfig-personal

# ...

By default, my name and email are always set to my personal identity. I also store some other global settings here, but those are not relevant for this post. If the repository is located inside the ~/work directory, a file named ~/.gitconfig-work is included. This is just another gitconfig file. This is what that looks like in my case:

[user]

name = Garrit Franke

signingkey = 12345678

email = garrit@work.de

[commit]

gpgsign = true

I hope you'll see where this is going. For every identity, you keep a separate .gitconfig file and include it in the main ~/.gitconfig. Crucially, this requires you to organize your repositories grouped by client.

=-=-=-=-=

Managing multiple Git identities involves more than just configuring user details. Here are some additional considerations:

**1. Git Version:**

Ensure that you have a recent version of Git installed to take advantage of the latest features and improvements.

bashCopy code

git --version

**2. Update Git Configurations:**

Regularly review and update your global and local Git configurations as needed. For instance, if your email address changes, update it in your Git configurations.

**3. Git Credential Manager:**

Depending on your operating system, you may need to configure a Git credential manager to securely store and retrieve credentials.

**4. SSH Key Permissions:**

Ensure that the permissions on your SSH private keys are secure (e.g., readable only by the owner):

bashCopy code

chmod 600 ~/.ssh/id\_rsa\_personal

chmod 600 ~/.ssh/id\_rsa\_work

**5. SSH Agent:**

Use an SSH agent to manage your keys and avoid entering your passphrase every time. Add your SSH private keys to the agent:

ssh-add ~/.ssh/id\_rsa\_personal

ssh-add ~/.ssh/id\_rsa\_work

**6. HTTPS vs. SSH:**

Decide whether to use HTTPS or SSH for cloning repositories. If you use HTTPS, you'll need to input credentials more often.

**7. Commit Signatures:**

If you sign your commits, ensure that your GPG key is correctly configured for each identity.

**8. Proxy Configuration:**

If you're behind a firewall or using a proxy, configure Git to work through the proxy.

**9. Git Ignore Files:**

Create and maintain **.gitignore** files for each project to exclude unnecessary files from version control.

**10. Documentation:**

Document your Git identity management process, especially if you are working in a team. Make it easy for others to understand and follow.

**11. Regular Backups:**

Backup important configuration files, especially if you use custom aliases or complex configurations.

**12. Review Branch Policies:**

For work-related projects, understand and adhere to any branch policies or workflows established by your team.

**13. Educate Team Members:**

If you are working in a team, educate team members on your Git identity management approach to avoid confusion.

**14. Review Repository URLs:**

Periodically review and update repository URLs in your configurations to ensure they are still valid.

**15. Git Hooks:**

Explore and use Git hooks if you need to automate certain tasks before or after specific Git actions.

By addressing these considerations, you can create a robust and efficient workflow for managing multiple Git identities. Regularly revisit your setup to adapt to changes in your projects or work environment.