

---

## Lab 1.1: Proposed Solution

---

### Install the Git package

---

#### Example for Debian/Ubuntu

```
$ apt-get install git
```

#### Example for RHEL/CentOS

```
$ yum install git
```

#### Example for Fedora

```
$ dnf install git
```

---

## Lab 1.2: Proposed Solution

---

### Install Git Bash Completion

---

#### Example for Debian/Ubuntu

```
$ apt-get install bash-completion
```

#### Example for RHEL/CentOS

```
$ yum install bash-completion
```

#### Example for Fedora

```
$ dnf install bash-completion
```

#### Fetch the git-prompt.sh script

```
$ wget https://raw.githubusercontent.com/git/git/master/contrib/completion/git-prompt.sh
```

#### Customize your prompt in your .bashrc file

```
$ vim $HOME/.bashrc  
source ~/git-prompt.sh  
export GIT_PS1_SHOWDIRTYSTATE=1  
export PS1='\u@\h] \W$(_git_ps1 " (%s)") \> '  
$ source $HOME/.bashrc
```

---

## Lab 2.1: Proposed Solution

---

Configure your username and email address

---

Set the global username

```
$ git config --global user.name "Michael Friedrich"
```

Set the global email address

```
$ git config --global user.email "michael.friedrich@netways.de"
```

Verification

```
$ git config --global --list
```

In addition to that you can open the `.gitconfig` file in your `$HOME` directory.

```
$ less $HOME/.gitconfig
```

Notes

You can also use `git config --global --list` to list all configured options.

---

## Lab 3.1: Proposed Solution

---

Clone an existing Git repository

---

Git clone

```
$ cd $HOME  
$ git clone https://github.com/Icinga/icinga2.git
```

---

## Lab 3.2: Proposed Solution

---

Initialize Git repository

---

Create a new Git repository

```
$ cd $HOME  
$ mkdir training  
$ cd training  
$ git init
```

---

## Lab 3.3: Proposed Solution

---

Add a new README.md file

---

### Example

```
$ cd $HOME/training  
$ echo "# Git Training Notes" > README.md  
$ git add README.md  
$ git status
```

---

## Lab 3.4: Proposed Solution

---

Remove file from staging index.

---

### Example

```
$ cd $HOME/training
$ git status
$ git reset --soft README.md
$ git status
$ git add README.md
$ git status
```

---

## Lab 3.5: Proposed Solution

---

### Examine current changes

---

#### Change files

```
$ cd $HOME/training
$ vim README.md
# Git Training Notes

I've learned about `git add` already.
...
git status
...
```

Modify and save the file.

#### Use git status

```
$ git status
```

You'll recognize the unstaged changes compared to your staging area.

#### Add the change to the staging area

```
$ git add README.md
```

#### Use git status again

```
$ git status
```

---

## Lab 3.6: Proposed Solution

---

### Play with Git Diff

---

#### Change files

```
$ vim README.md
```

```
I've also learned the difference between local changes and the staging area.
```

Modify and save the file.

#### Use git diff

```
$ git diff
```

You'll recognize the unstaged changes compared to your staging area.

#### Add the change to the staging area

```
$ git add README.md
```

#### Use git diff again

```
$ git diff
```

---

## Lab 3.7: Proposed Solution

---

### Add .gitignore file and exclude files/directories

---

#### Add file/directory

Create a dummy file and directory which contains a file itself. This is for simulating unwanted files in the working directory.

```
$ cd $HOME/training
$ touch generated.tmp
$ mkdir debug
$ touch debug/timestamp
```

#### Examine the state with git status

```
$ git status
On branch master

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  debug/
  generated.tmp
```

#### Add .gitignore file

```
$ vim .gitignore
*.tmp
debug/
```

Files matching the `*.tmp` pattern in the current directory will be excluded. Furthermore the `debug` directory (note the trailing slash).

#### Examine the state with git status

```
$ git status
On branch master

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  .gitignore
```

We'll learn how to add and commit the untracked `.gitignore` file in later examples.

---

## Lab 4.1: Proposed Solution

---

### Commit Changes

---

#### Add/modify file

```
$ cd $HOME/training
$ vim README.md
`git commit` also has `-a` which should be used with care.
$ git add README.md
```

#### Commit the changed file

```
$ git status
$ git commit -v README.md
Update training notes

My first commit :)

Save and exit.
```

#### Verify the Git history

```
$ git log
```

---

## Lab 4.2: Proposed Solution

---

### Examine the Commit History

---

#### Add and commit remaining changes

```
$ cd $HOME/training
$ git status
$ git add .gitignore
$ git commit -v .gitignore -m "Add .gitignore file"
```

#### Use git log

```
$ git log
```

#### Use git show

```
$ git show
$ git show <commitid>
$ git show -2
```

#### Use git diff

```
$ git diff
$ git diff <commitid1> <commitid2>
```

---

## Lab 4.3: Proposed Solution

---

### Learn more about tig

---

#### Install tig

CentOS 7 (requires the EPEL repository):

```
# yum install epel-release  
# yum makecache  
# yum install tig
```

Debian/Ubuntu

```
# apt-get install tig
```

#### Use tig

```
$ cd $HOME/training  
$ tig  
  
@@@ Sh  
$ cd $HOME/icinga2  
$ tig
```

---

## Lab 4.4: Proposed Solution

---

### Amend changes to commits

---

#### Add the files

```
$ cd $HOME/training
$ vim README.md
`git commit --amend` adds missing changes to the previous commit.
```

#### Add README.md to the staging index and commit the change

```
$ git add README.md
$ git status
$ git commit -v README.md -m "Add documentation for Amend"
```

#### Add a note to README.md again

```
$ vim README.md
`git commit --amend` adds missing changes to the previous commit.
I'm also able to edit the commit message.
$ git add README.md
$ git status
```

#### Amend README.md to the previous commit

```
$ git commit --amend -v README.md
Add documentation for 'git amend'
```

Adopt the commit message as additional exercise above.

```
$ git status
$ git show
```

---

## Lab 5.1: Proposed Solution

---

Show the current branch

---

### Example

```
$ cd $HOME/training  
$ git branch  
master
```

---

## Lab 5.2: Proposed Solution

---

### Create and checkout a new branch

---

#### Create the branch

```
$ cd $HOME/training
$ git branch feature/docs master
```

#### List the branches

```
$ git branch
master
feature/docs
```

#### Checkout the created branch

```
$ git checkout feature/docs
$ git branch
master
feature/docs
```

#### Use it all at once

`git checkout -b` creates a new branch from the current branch and does the checkout afterwards. That way you'll save some time when working with branches quite often.

```
$ git checkout master
$ git checkout -b feature/docs2
```

---

## Lab 5.3: Proposed Solution

---

Delete the branch

---

Checkout the master branch

```
$ cd $HOME/training  
$ git checkout master
```

Delete the previously created branch

```
$ git branch -d feature/docs2
```

Try to delete the current branch

```
$ git checkout master  
$ git branch -d master
```

---

## Lab 5.4: Proposed Solution

---

Show the second commit

---

### Example

```
$ cd $HOME/training  
@@@ Sh  
$ git show HEAD^
```

or

```
$ git show HEAD~1
```

---

## Lab 5.5: Proposed Solution

---

### Show history of different branch

---

#### Create a new branch

```
$ git checkout master  
$ git checkout -b feature/dpcs
```

#### Switch to the master branch

```
$ git checkout master
```

#### Show the history of the other branch

```
$ git show feature/docs
```

#### Commit change on master

```
$ echo "I understand HEAD now" >> README.md  
$ git add README.md  
$ git commit -v README.md -m "What I've learned so far: HEAD"
```

#### Show diff between HEAD and branch

```
$ git diff HEAD feature/docs
```

---

## Lab 6.1: Proposed Solution

---

### Create a new GitLab app in NWS

---

Open <https://nws.netways.de> in your browser and register your personal account. Login and navigate to Apps > GitLab CE > Basic and deploy the app.

Use Live View and set a secure root password. The login into GitLab.

---

## Lab 6.2: Proposed Solution

---

Create a new GitLab project for the current user

---

### Add Project

GitLab 10.x provides `New Project` underneath the `+` icon in the top menu bar, next to the search form.

Fill in the `Project name` form with `training` and leave the other options as default.

### Project View

You'll notice the `HTTPS` URL centered below the project name.

We will be using this remote URL for connecting our local repository in the next step.

Right now the repository is empty and does not contain any file. GitLab offers you to add new files, e.g. a `README.md` file or `LICENSE` details directly in the browser. In the background, it is still committing the changes to the Git repository.

---

## Lab 6.3: Proposed Solution

---

### Configure client credentials helper

---

#### Verify and set

```
$ git config --global --get credential.helper  
/usr/libexec/git-core/git-credential-gnome-keyring
```

If not set, you can set it explicitly.

```
$ git config --global credential.helper /usr/libexec/git-core/git-credential-gnome-keyring
```

---

## Lab 6.4: Proposed Solution

---

### Add the repository as remote origin

---

#### Add the remote origin

```
$ cd $HOME/training.git
$ git remote add origin https://[...].nws.netways.de/root/training.git
$ git fetch
```

#### Push the history

```
$ git push
```

This will not work since the local branch does not follow the remote branch. Use `--set-upstream` as proposed by the cli output. Short form is `-u`.

```
$ git push --set-upstream origin master
```

#### Set default push method

Git versions prior 2.0 did not define the default push method. The default behaviour was to use the same local branch name for the remote branch name.

The new default method should be `simple` which ensures that the local branches will only be pushed to remote branches which `git pull` is following.

Our setup did not clone the repository (which includes a virtual git pull). Therefore the local master branch does not follow a remote branch.

In order to fix that, add the default push method to your global configuration.

```
git config --global push.default simple
```

---

## Lab 6.5: Proposed Solution

---

Examine GitLab's project history

---

### Project History

Choose `History` and look at the Git commits, their author, subject and timestamp.

Compare it with the local `git log` or `tig` entries.

---

## Lab 7.1: Proposed Solution

---

Learn more about git push

---

### Make changes

```
$ cd $HOME/training  
$ git checkout master  
  
$ vim README.md
```

Now I know how to publish my changes to my NWS hosted GitLab server.

```
$ git add README.md  
$ git commit -v README.md -m "Add docs for git push"
```

### Push changes

```
$ git push origin master
```

---

## Lab 7.2: Proposed Solution

---

Learn more about git fetch and git pull

---

Reset your local commit history by one

```
$ git log  
$ git reset --hard HEAD~  
$ git log
```

Fetch changes

```
$ git fetch
```

Compare this with your local commit history - you'll see that there are not changes pulled yet.

```
$ git log
```

Pull changes

```
$ git pull
```

Check the local commit history - now your local history has been updated with the remote history.

```
$ git log
```

---

## Lab 7.3: Proposed Solution

---

### Add git tag

---

#### Add tag

```
$ git log  
$ git tag -m "Release v0.1" v0.1
```

#### Verify tag

```
$ git tag -l
```

#### Push tags to remote origin

```
$ git push --tags
```

---

## Lab 7.4: Proposed Solution

---

### Learn more about git stash

---

#### Edit README and add docs

```
$ cd $HOME/training
$ vim README.md
```

Now I am learning how to use git stash and temporarily drop the changes  
 e.g. to change into another branch.

#### Examine the state with git status

```
$ git status
```

#### Stash changes

```
$ git stash
Saved working directory and index state WIP on master: 31dcde5 Add docs for git push
HEAD is now at 31dcde5 Add docs for git push
```

#### Examine the state with git status

```
$ git status
On branch master
nothing to commit, working tree clean
```

#### Examine the stash list

```
$ git stash list
stash@{0}: WIP on master: 31dcde5 Add docs for git push

@@@ Sh
$ git stash show -p

diff --git a/README.md b/README.md
index 2081a37..550db95 100644
--- a/README.md
+++ b/README.md
@@@ -15,3 +15,7 @@@ Now for real.
...

git commit --amend
...

+
+## Git Stash
+
+`git stash`
```

#### Fetch previously stashed changes

```
$ git stash pop
```

```
Dropped refs/stash@{0} (a9f28340e6d536a9179307bd26169368e450161f)
```

---

## Lab 8.1: Proposed Solution

---

Clone the training repository twice and add diverging commits

---

### Clone the central repository

The last parameter specifies the target directory `training1` .

```
$ cd $HOME  
$ git clone https://[...].nws.netways.de/root/training.git training1
```

### Add/modify a file

```
$ echo "My Name" > AUTHORS
```

### Commit the change

```
$ git add -A .  
$ git commit -av -m "Add myself to AUTHORS"
```

### Push the changes to the remote repository

```
$ git push origin master
```

### Repeat these steps with training2

The last parameter specifies the target directory `training2` .

```
$ cd $HOME  
$ git clone https://[...].nws.netways.de/root/training.git training2
```

---

## Lab 8.2: Proposed Solution

---

### Rebase your local history with the remote repository

---

#### Reset the history by 2 commits

```
$ git reset --hard HEAD~2
```

#### Update and commit README.md

```
$ vim README.md
Testing rebase
$ git add README.md
$ git commit -v README.md "Rebase test"
```

#### Fetch and diff the remote changes

```
$ git fetch
$ git diff origin/master
```

#### Rebase your local history

Rebase your local history against the remote origin master branch.

```
$ git rebase origin/master
```

#### Resolve merge problems

```
$ git status
$ vim README.md
```

Search for conflicts in vim:

```
/>>>
```

Resolve the conflicts, add the file and continue the rebase.

```
$ git add README.md
$ git rebase --continue
```

#### Push the changes to the remote repository

```
$ git push origin master
```

---

## Lab 8.3: Proposed Solution

---

### Use Feature Branches

---

#### Create a new branch

The new branch `feature/docs-workflows` will be based on the `master` branch.

```
$ cd $HOME/training
$ git checkout master
$ git checkout -b feature/docs-workflows
```

#### Add and commit changes

```
$ vim README.md
## Workflows

Central Workflow and now feature workflows with descriptive branch names.
$ git add README.md
$ git commit -v README.md -m "Update docs for Git workflows"
```

#### Push your feature branch

```
$ git push -u origin feature/docs-workflows
```

---

## Lab 8.4: Proposed Solution

---

### Merge Feature Branches

---

Checkout the feature branch and add a commit

```
$ cd $HOME/training
$ git checkout feature/docs-workflows
$ vim README.md

I'm learning about workflows today.

$ git add README.md
$ git commit -v README.md -m "Update docs for workflows"
$ git push origin feature/docs-workflows
```

Checkout the feature branch and compare it with the master branch

```
$ git branch
$ git checkout feature/docs-workflows
$ git diff master feature/docs
```

Checkout the master and merge the feature branch

```
$ git checkout master
$ git merge --no-ff feature/docs-workflows

In this commit message, I may add a reference to a GitLab issue like this
to automatically resolve it after merge.

fixes #12
```

Examine the history

```
$ tig
```

## Lab 8.5: Proposed Solution

### Merge Feature Branches

Checkout the feature branch, add, commit and push changes

```
$ cd $HOME/training
$ git checkout master
$ git checkout -b feature/docs-merge-request
$ vim README.md

Let's create a merge request with GitLab.

$ git add README.md
$ git commit -v README.md -m "Update docs for merge requests"
$ git push -u origin feature/docs-merge-request
```

Navigate into GitLab and create merge request

GitLab puts the URL into the shell output on pushing the branch.

```
$ git push -u origin feature/docs-merge-request
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 618 bytes | 618.00 KiB/s, done.
Total 6 (delta 4), reused 0 (delta 0)
remote:
remote: To create a merge request for feature/docs-merge-request, visit:
remote: https://[...].nws.netways.de/root/training/merge_requests/new?merge_request%5Bsource_branch%5D=feature%2Fdocs-merge-request
remote:
To https://[...].nws.netways.de/root/training.git
 * [new branch] feature/docs-merge-request -> feature/docs-merge-request
Branch 'feature/docs-merge-request' set up to track remote branch 'feature/docs-merge-request' from 'origin'.
```

Open the URL in your browser.

Specify a topic, description and create the merge request.

Merge the request to master and investigate the Git history.

Pull changes to master after merge

```
$ git checkout master
$ git pull
$ tig
```

---

## Lab 8.6: Proposed Solution

---

### Rebase and squash commits

---

#### Add 3 commits to your history

If you do not have any.

```
$ git checkout master
$ git checkout -b feature/squash-test

@@@ Sh
$ echo "# Rebase and Squash" >> README.md
$ git commit -av -m "commit1"
$ echo " " >> README.md
$ git commit -av -m "commit2"
$ echo "`git rebase -i` is interactive" >> README.md
$ git commit -av -m "commit3"
```

#### Use git rebase to squash three commits

Use the interactive mode by specifying the `-i` parameter. `HEAD~3` uses the commit range three commits to the current HEAD commit.

```
$ git rebase -i HEAD~3
```

The interactive mode opens the editor you are familiar with from commit messages.

#### Choose the commits to squash

```
pick 5a31d9e commit1
squash ce90e16 commit2
squash ed6a68f commit3
```

---

## Lab 8.7: Proposed Solution

---

### Delete remote branch

---

#### Create and push remote branch

If you do not have any.

```
$ cd $HOME/training
$ git checkout master
$ git checkout -b feature/docs-wrong-name
$ git push -u origin feature/docs-wrong-name
```

#### Identify remote branch to delete

```
$ git branch -r
feature/docs-wrong-name
```

#### Delete remote branch

```
$ git push origin :feature/docs-wrong-name
```

Now verify it is gone (Hint: `-r` lists remote branches).

```
$ git fetch
$ git branch -r
```

## Lab 10.1: Proposed Solution

### Open Runner Administration View

Navigate to `/admin/runners` .

How to setup a shared Runner for a new project Install a Runner compatible with GitLab CI (checkout the GitLab Runner section for information on how to install it). Specify the following URL during the Runner setup: ... Use the following registration token during setup: ... Start the Runner!

#### Runners

Registered runners are listed at the bottom.

#### Register Runner

- Steps:
  - Run `gitlab-runner register` as root
  - Use the HTTP Url as host
  - Paste the token
  - Add description `training01` and tag `training`
  - Untagged builds: `true` , Lock to current project: `false`
  - Executor: `docker` , Default: `alpine/latest`

Reference: <https://gitlab.com/gitlab-org/gitlab-runner/blob/master/docs/install/linux-repository.md>

Reference: <https://docs.gitlab.com/runner/install/linux-repository.html>

Reference for Docker: <https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/>

Example on Ubuntu:

```
`` apt-get install \ apt-transport-https \ ca-certificates \ curl \ software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
add-apt-repository \ "deb [arch=amd64] https://download.docker.com/linux/ubuntu \ $(lsb_release -cs) \
stable"
apt-get update apt-get install docker-ce ``
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh | sudo bash apt-get install gitlab-
runner
```

Start CLI

```
# gitlab-runner register
Running in system-mode.

Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
http://192.168.56.101

Please enter the gitlab-ci token for this runner:
Do1aqTvPiiBj6_u5Ye

Please enter the gitlab-ci description for this runner:
```

[ubuntu-16]: training01

Please enter the gitlab-ci tags **for** this runner (comma separated):  
training

Whether to run untagged builds [true/false]:  
[false]: true

Whether to lock the Runner to current project [true/false]:  
[true]: false

Registering runner... succeeded runner=Do1aqTvP

Please enter the executor: docker+machine, docker-ssh, parallels, ssh, virtualbox, docker, shell, docker-ssh+machine, kubernetes:  
docker

Please enter the default Docker image (e.g. ruby:2.1):  
alpine:latest

Runner registered successfully. Feel free to start it, but **if it's running already the config should be automatically reloaded!**

---

## Lab 10.2: Proposed Solution

---

### Create .gitlab-ci.yml file

---

### Create CI configuration file

```
$ cd $HOME/training
$ vim .gitlab-ci.yml

image: alpine:latest

all_tests:
  script:
    - exit 1
```

The script will always fail. We will use different exit states to fix it. Future examples and tests work the same way.

---

## Lab 10.3: Proposed Solution

---

### Push CI config and trigger GitLab job

---

#### Git Add, Commit, Push

```
$ git add .gitlab-ci.yml  
$ git commit -av -m "Add GitLab CI config"  
$ git push origin master
```

#### Modify exit code

```
$ vim .gitlab-ci.yml  
  
image: alpine:latest  
  
all_tests:  
  script:  
    - exit 0
```

---

## Lab 10.4: Proposed Solution

---

### Prepare container to convert Markdown to HTML

---

#### Edit `.gitlab-ci.yml` and add `before_script`

```
$ vim .gitlab-ci.yml
image: alpine:latest
before_script:
```

#### Update apk and install Python/pip

```
$ vim .gitlab-ci.yml
image: alpine:latest
before_script:
- apk update && apk add python py-pip
```

#### Install Markdown Python libraries

```
$ vim .gitlab-ci.yml
image: alpine:latest
before_script:
- apk update && apk add python py-pip
- pip install markdown Pygments
```

#### Verify the content

```
$ vim .gitlab-ci.yml
image: alpine:latest
before_script:
- apk update && apk add python py-pip
- pip install markdown Pygments
all_tests:
script:
- exit 0
```

#### Commit and push the changes

```
$ git commit -av -m "CI: Prepare markdown conversion"
$ git push
```

---

## Lab 10.5: Proposed Solution

---

### Create HTML docs from Markdown

---

Edit `.gitlab-ci.yml` and add markdown section

```
$ vim .gitlab-ci.yml
...
all_tests:
  script:
    - exit 0
markdown:
```

Add script to convert Markdown into HTML

```
$ vim .gitlab-ci.yml
...
markdown:
  - python -m markdown README.md > README.html
```

Store artifacts

Add `paths` section which includes `README.html` as entry. Tell GitLab to expire this artifact in `1 week` .

```
$ vim .gitlab-ci.yml
...
markdown:
  - python -m markdown README.md > README.html
artifacts:
  paths:
    - README.html
  expire_in: 1 week
```

Verify the content

```
$ vim .gitlab-ci.yml
image: alpine:latest

before_script:
  - apk update && apk add python py-pip
  - pip install markdown Pygments

all_tests:
  script:
    - exit 0
markdown:
```

```
- python -m markdown README.md > README.html
artifacts:
  paths:
  - README.html
  expire_in: 1 week
```

## Commit and push the changes

```
$ git commit -av -m "CI: Generate HTML docs from Markdown"
$ git push
```

## Download HTML artifacts

Navigate into the repository > `CI / CD` > `Jobs` > `#...` and choose `Job Artifacts`. Download them, extract them and open the HTML file with your browser.

---

## Lab 10.6: Proposed Solution

---

### Update docs

---

#### Edit README.md and add learned details

```
vim README.md  
  
# CI Runners  
  
....
```

#### Commit and push changes

```
git commit -av -m "Add notes on CI runners"  
git push
```

#### Download HTML artifacts

Navigate into the repository > CI / CD > Jobs > #... and choose Job Artifacts. Download them, extract the and open the HTML file with your browser.

---

## Lab 11.1: Proposed Solution

---

### Use Git Blame

---

Pick a file and use Git Blame

```
$ git blame README.md
```

Modify and commit changes and use Git Blame again

```
$ echo "Blame me" >> README.md && git commit -av -m "Blame me"  
$ echo "Blame me, too." >> README.md && git commit -av -m "Blame me, too"  
$ git blame README.md
```

---

## Lab 11.2: Proposed Solution

---

Add an alias for git diff

---

Edit the git configuration file

```
$ vim $HOME/.gitconfig  
[alias]  
d = diff
```

Make a change and test it

```
$ vim AUTHORS  
:dd  
$ git d
```

