Instructions for Server Usage

Host `naf.cs.hut.fi` is our course server for completing assignments 2 and 3. It runs a standard Ubuntu Linux with some basic tools installed. If the server lacks some tools that you need, please send an email to the course staff. We will also have some reception sessions for resolving server-related issues. Nevertheless, please focus on the assignment and not debugging the server. If you need any help, please let us know.

WARNING: Please make your applications secure enough to prevent at least simplest vulnerabilities such as **unchecked user input** in database queries or file paths or eating up all the disk space with senseless logging. Keep in mind that you are working in a live server environment that is accessible with HTTP from the Internet.

Software

- PHP 5.3
- Python 2.7 (default) and 3.2
    - Django 1.5
    - Flask 0.9
- Ruby 1.9.3
    - Rails 3.2.12
- Version control tools: git, bzr, hg, svn

User Accounts

Every student group in the course has only one user account, but with a different login name and password for each member. The home directory has permissions that prevent anyone else but you, course staff and some admins from seeing the files in it. Course staff may access your home directory to help you with a problem, monitor progress of the course or evaluate work.

You may login with SSH (at least) from the university network. If you are outside the university network, connect with your favourite SSH client first to `kosh.aalto.fi` (using Aalto user account) or `ukk.cs.hut.fi` (using Niksula user account) and then `ssh` to `naf.cs.hut.fi`. Remember to specify your username, e.g. `ssh mmeikal@naf.cs.hut.fi`.

Your username: u + your student number (letter in lowercase); e.g. **`u12345a`**

Your password: provided to you in a separate email (you may always use an SSH key if you want, just append your public key to `~/.ssh/authorized_keys`)

Synchronizing Your Work Using Git

We recommend that if you wish to have a "production" version of your project, you would use Git to synchronize it with your development work. For this, you need to set up a deployment repository. In addition, you may want to have a work repository for each of the group members. These repositories would then be synchronized with an additional repository, bringing the total count of repositories with a team of two students to four.

The instructions below are supposed to give you a hint, how you should configure your Git to distribute your work efficiently. If you feel that you don't understand them and run into troubles, please send us an email.

Setting up a remote deployment repository

This alternative assumes that you do development work at a desktop computer, then push it into a remote repository and want to pull your work from the remote repository to the server for deployment.

Create another SSH key that will be used for synchronizing the work by simply running `ssh-keygen` at the server. Since all the work you do at the server is bound to the user account that is created only for the purposes of this course, no more complex configuration (e.g. repository-specific keys) is necessary. Next, you will need to add the public key to the access configuration of your project repository. GitLab and many other services allow adding so-called deployment keys for repositories, which you specifically use for such deployment purposes. They allow a read-only access to your repository in GitLab. On the other hand, you may also choose to add the public key into your personal profile, giving the deployment repository the same privileges that you have.

If you are using only the GitLab project provided by the course and wish to have a read-only deployment key, please send the public key to us in an email and we will add it into your project. If you have a separate project in GitLab or elsewhere, you should be able to configure it by yourself.

Development work at the server

If you want to do some development at the server and thus commit some code, please note that a separate repository for each student is recommended as development work is a personal thing. In other words, please clone your own copy into the directory that is easily recognizable to be only yours (e.g. `myname-repo`). Your partner may then make another clone for a personal working copy. In no circumstances it is advisable to do development with one and only working copy, which many persons would use simultaneously. Also make sure to configure repository-specific name and email (`git config user.(name|email)`).

In order to synchronize the code, you will also need at least one SSH key that allows both reading and writing to your project repository to make upstream pulls/pushes. To this course, it does not really matter if you share one key for your personal repositories and the deployment repository. If you have some other repositories related to the same user account in the project management service (e.g. GitLab) that you are using and don't want to share your other projects with your partner, try creating

a new SSH key with a passphrase, configure Git to use it with the remote of your personal repository and enter the passphrase whenever you pull/push.

To avoid playing with SSH keys or logins altogether, you may always have your own synchronization repository in your home folder. Initialize it with flag `--bare` (no working copy will be created), then add it as a remote of your development repository (file path suffices as the address). You may then clone some more repositories from this repository. As there is no additional access management, this is the easiest way. It is then up to you to finally submit the work to the repository of the course.

Workflow for deployment-specific settings

It is quite usual to have some deployment-specific settings that should be kept separately from the project especially if there are many deployments. For instance, such a scenario for this course would be to have personal development environments for the both students of a group in addition to the demonstration deployment at the server. Thus there would be three different configurations.

One way to keep configurations safe and separate is not to put the related files under version control at all. Then you may just add them to `.gitignore` to make Git ignore them in `status`. Sometimes, however, these need to be versioned as well (e.g. when configuration parameter set changes over time). One way to do this is to create another branch at the deployment repository (let's call it `deployment`) and keep it checked out. Everytime you'll pull the changes of the project repository to the `master` branch, you would also then merge the changes of `master` into your `deployment` branch.

Ways to Serve Your Website

One Apache process group serves everyone. However, all your own code will have the same privileges as you have in the shell. If you want to run your application on another web server, see `Apache Rewrite as Proxy`.

Web Content

There is a single virtual host for each of the groups. The base URL **`http://groupXX.naf.cs.hut.fi/`** is mapped to `public_html` in your home directory. For every directory under `public_html`, Apache searches by default for `index.html`, `index.fcgi`, `index.php` and `index.cgi` (in this order). Your home directory contains a symbolic link to the HTTP log files directory of your virtual host. HTTPS is also available, but the signature is self-signed.

You may modify some of Apache's settings through `.htaccess` files. A couple of examples follow:

```
# this is a .htaccess file

# allowing directory listing
Options +Indexes

# limiting access to the server machine only
Order deny,allow
Deny from all
Allow from 127.0.0.1
```

CGI and FastCGI

CGI is a simple way to direct the HTTP request to a fresh process (note that some headers in the program output are necessary), but it is rather slow due to overhead from starting up the process. In comparison, FCGI keeps the process running after first request but requires a program crafted for the purpose with an incoming event handler.

You will probably use a readily-available script that initialises your favourite framework: this is the case with the following Django example. However, you may run your own programs as well. If they are to be interpreted, define the interpreter on the first line of the entry-point script. For instance, with Python 2.7 you would use:
`#!/usr/bin/python`

By default, Apache considers the files with following suffixes as CGI or FCGI programs:

- CGI: `.cgi` files
- FCGI `.fcgi` files; idle timeout is set to 10 minutes

Whenever you use CGI or FCGI, remember to set execute bit on for your program file. Otherwise it will not run. In addition, you may always kill your existing processes when you want to reload your application so that you don't need to wait for the timeout.

```
# Setting execute bit for a file
chmod a+x filename

# Killing a process
# 1) find the id of the process you want to kill
# 2) send SIGTERM the process;
#    if the process is really stuck,
```

```
#    you may want to try out flag "-9" (SIGKILL)
ps a
kill process_id
```

Apache Rewrite as Proxy

If you find Apache's script modules too cumbersome to work with, you may use mod_rewrite as a proxy to run your own server processes. With the following code clip, you can forward all the HTTP requests that have a certain path to your own HTTP server that is running on the same machine. The remaining tail of the address will be requested from your local server.

`.htaccess`:

```
RewriteEngine on
# replace PORTNUMBER with a random upper-range port number of your choice
RewriteRule ^(.*) http://localhost:PORTNUMBER/$1 [P]
```

PHP

PHP works out of the box with help of mod_suphp. Just put your code to .php files into `public_html`. Of course, you may also run PHP application as a (F)CGI script.

Perl

FastCGI library is installed for Perl, but we will not debug your Perl code.

Python / Django

Python and Django are ready for use. You may also use any Python software with appropriate (F)CGI scripts. For Django, we also describe here a way to use Django's internal development server. Here are the minimalistic tutorials for usage, but for more information please refer to Django documentation.

Creating Django Project

```
# This should not be done in your public_html!
# Create a project. This will create a directory "project_name"
django-admin.py startproject project_name
```

Method 1: Serving Django Project with Development Server

1. Make a new directory to public_html. You can decide the name.
2. Follow the instructions in the section about rewriting for proxy to create a *.htaccess* to the new directory.
3. Go to Django project directory.
4. Type the following command:

   ```
   python manage.py runserver PORTNUMBER
   ```

5. Navigate with your browser to the URL corresponding the directory you created.

Method 2: Serving Django Project with FastCGI

1. Make a new directory to `public_html`. You can decide the name.
2. Create a new file `index.fcgi` with the following contents.
3. Navigate with your browser to the URL corresponding the directory you created.

`index.fcgi`:

```
#!/usr/bin/python
import sys, os
sys.path.insert(0, "absolute path to Django project")

# Set the DJANGO_SETTINGS_MODULE environment variable.
os.environ['DJANGO_SETTINGS_MODULE'] = "project_name.settings"

from django.core.servers.fastcgi import runfastcgi
runfastcgi(method="threaded", daemonize="false")
```

Python / Flask

Flask is a mini-framework for web development. For those who want to use Flask, please go ahead, but please also note that unlike Django it does not have any active record -style object-relational mapper (ORM). We have installed the base package (and dependencies) for testing it out.

Python / Other Software

If you want to install lots of Python software, we recommend you to get familiar with virtualenv at **http://www.virtualenv.org/**. It will allow you to create your own virtual environment and install software with `pip` as you like. This is helpful also if you need a specific version of software, such as a 1.3 version of Django (`pip install django==1.3.7`).

Ruby / RoR

Installing Ruby on Rails on the server is really easy, just run:

```
\curl -L https://get.rvm.io | bash -s stable --rails
```

To start using RVM you need to run

```
source ~/.rvm/scripts/rvm
```

For your convenience please add this line also to the configuration file of your favourtite shell:

Bash: `echo "source ~/.rvm/scripts/rvm" >> ~/.bashrc`

Zsh: `echo "source ~/.rvm/scripts/rvm" >> ~/.zshrc`

To start using rails you need to run

```
rails new <project_dir>
```

Ruby / Sinatra and other Ruby software

If you are not keen on using Ruby on Rails, but you would use like to use other Ruby software then follow the instructions from above but replace the rvm install command with:

```
\curl -L https://get.rvm.io | bash -s stable --ruby
```

And then install your favorite gems with, e.g.:

```
gem install sinatra
```

Other Services

Databases

MySQL and PostgreSQL databases are available on request, but we may also be able to provide you some other database (e.g. a NoSQL database). As database performance is at very low priority, we however recommend as simple solutions as practical.

Scheduled Tasks

If you need some scheduled scripts, you might want to use `cron`. With `crontab -e` you can edit the list and settings of your own scheduled tasks. For more information, Wikipedia serves as a good starting point.