First-Time Git Setup

Now that you have Git on your system, you'll want to do a few things to customize your Git environment. You should have to do these things only once; they'll stick around between upgrades. You can also change them at any time by running through the commands again.

Git comes with a tool called git config that lets you get and set configuration variables that control all aspects of how Git looks and operates. These variables can be stored in three different places:

- `/etc/gitconfig` file: Contains values for every user on the system and all their repositories. If you pass the option `--system` to `git config`, it reads and writes from this file specifically.
- `~/.gitconfig` file: Specific to your user. You can make Git read and write to this file specifically by passing the `--global` option.
- config file in the git directory (that is, `.git/config`) of whatever repository you're currently using: Specific to that single repository. Each level overrides values in the previous level, so values in `.git/config` trump those in `/etc/gitconfig`.

On Windows systems, Git looks for the `.gitconfig` file in the `$HOME` directory (`%USERPROFILE%` in Windows' environment), which is `C:\Documents and Settings\$USER` or `C:\Users\$USER` for most people, depending on version (`$USER` is `%USERNAME%` in Windows' environment). It also still looks for /etc/gitconfig, although it's relative to the MSys root, which is wherever you decide to install Git on your Windows system when you run the installer.

Your Identity

The first thing you should do when you install Git is to set your user name and e-mail address. This is important because every Git commit uses this information, and it's immutably baked into the commits you pass around:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Again, you need to do this only once if you pass the `--global` option, because then Git will always use that information for anything you do on that system. If you want to override this with a different name or e-mail address for specific projects, you can run the command without the `--global` option when you're in that project.

Your Editor

Now that your identity is set up, you can configure the default text editor that will be used when Git needs you to type in a message. By default, Git uses your system's default editor, which is generally Vi or Vim. If you want to use a different text editor, such as Emacs, you can do the following:

```
$ git config --global core.editor emacs
```

Your Diff Tool

Another useful option you may want to configure is the default diff tool to use to resolve merge conflicts. Say you want to use vimdiff:

```
$ git config --global merge.tool vimdiff
```

Git accepts kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge, and opendiff as valid merge tools. You can also set up a custom tool; see Chapter 7 for more information about doing that.

Checking Your Settings

If you want to check your settings, you can use the `git config --list` command to list all the settings Git can find at that point:

```
$ git config --list
user.name=Scott Chacon
user.email=schacon@gmail.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

You may see keys more than once, because Git reads the same key from different files (`/etc/gitconfig` and `~/.gitconfig`, for example). In this case, Git uses the last value for each unique key it sees.

You can also check what Git thinks a specific key's value is by typing `git config {key}`:

```
$ git config user.name
Scott Chacon
```

```
adduser --disabled-login git
```

Vednuj!!
ssh git@192.168.0.85 mkdir -p .ssh

Pri vsiako addvane na PC

```
ssh-keygen -t rsa
```

cat .ssh/id_rsa.pub | ssh git@87.121.165.3 'cat >> .ssh/authorized_keys'

CLIENT:
ssh-add

SERVER:
apt-get install vim

```
git config --system user.name "NAME"
git config --system user.email johndoe@example.com
git config --system core.editor vim
git config --global merge.tool vimdiff
git config --list
git status
```

cd /home/git
mkdir <project>
cd <project>
git init
git --bare init

You can simply convert your remote repository to bare repository (there is no working copy in the bare repository - the folder contains only the actual repository data).
Execute the following command in your remote repository folder:

```
git config —-bool core.bare true
```

Then delete all the files except `.git` in that folder. And then you will be able to perform `git push` to the remote repository without any errors.

CLIENT:
git push -v --force --all ssh://git@87.121.165.3/home/git/testprj

vim README
git add README
git commit