Department of Computer Science
University of Pretoria

Software Engineering
COS301

Peer Introduction to Software Engineering Concepts

Broekman, A (Andrew)
U11089777

Congratultions, if you are reading this, it means that you have successfully cloned the Team Echo git repository. This document introduces you to some software concepts necessary for completion of the COS 301 mini project. In the process you will gain valuable skills to be used in industry and academia, setting you apart from fellow colleagues. This document provides you with practical tasks to complete, and you are urged to complete these tasks, as we know, the only way to learn in programming is to DIY (Do It Yourself).

1 Git

1.1 Git

Git is a distributed version control system or so called source code management system developed by Linus Torvalds in 2006 for managing the Linux kernel source code. Git is distributed, meaning that each developer has the complete repository history, independent of network access. This means, that even when you are offline, you still work on your code. It is important to note that Git and Github and also Git and Bitbucket are not the same. Git is the client used on your machine to manage your repository and keep a record of what changed in your code. Github and Bitbucket are services provided to the internet, to allow people to upload there Git repository onto a public server. Thus, you can view Git similar to Photoshop, you create your own graphic files and keep them on your local PC. However when you want to share your files with friends, you may want to upload your files to a cloud service such as Dropbox, Google Drive, Microsoft OneDrive etc.

1.2 Installing git on Ubuntu

To install the git client on Ubuntu, one would use the following commands in a terminal

```
# sudo apt-get update
# sudo apt-get install git
# git config --global user.name "Your Name"
# git config --global user.email "youremail@domain.com"
```

1.3 Installing git on Windows

For installation of git on Microsoft Windows, the easiest is to use one of the following software packages

1. Github for Windows
   This is a CLI and GUI git software package released by Github.com. More information is available at `https://desktop.github.com/`
2. Offical Git Client for Windows
   This is the offical Git client for Windows. More information is available at `https://git-scm.com/downloads`

Replace **Your Name** and **youremail@domain.com** with your own name and email address you used to register on Github, making sure to surround both in quotations.

1.4 Introduction to Git

For this part of the tutorial we will be using the git terminal, as this will allow you to understand the concepts much better.

Create a new directory called **cos301tutorial** in your home directory.

```
# cd ~
# mkdir cos301tutorial
# cd cos301tutorial
```

We will now create a new empty git repository in the **cos301tutorial** directory.

```
# git init
```

Now the real fun begins. Create a new file, called **index.html**, adding only the html opening and closing tags. Your index.html file should now look like this

```
<html>

</html>
```

In the terminal, issue the following command,

```
# git status
```

You should receive output similar to the following:

```
On branch master

Initial commit

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

        index.html

nothing added to commit but untracked files present (use "git␣add" to track)
```

Notice that git indicates that the index.html file is not tracked. This means that any changes you make, will not be recorded in the git history. Therefore when you create a new file, which should be tracked in git, it is important to ensure you add this file to git.

Do note, that we never commit and file that can be regenerated from other commited objects in the repository. Examples of such files are compiled class files, object files, executables, compiled pdf documents, auxiliary and log documents from TEX or LATEX compilation. This is the reason for the .gitignore file, which you are encourage to view and read more about, as any developer worth there salt will make use of a .gitignore file. A nice Git repository exists with gitignore files for multiple tools, languages, OS's etc., which assists developers in ensuring only the needed files make it into the repository. For more information refer to `https://github.com/github/gitignore`

To add this new file to git we use the command,

```
# git add index.html.
```

This now brings us to the three sections of a git project, the working directory, staging area, and Git directory as show in figure 1.

The Git directory, normally stored as a hidden directory, .git, in your root directory of your project, is the database of your repository. Git stores all files, also know in the git world as objects, as well as metadata in this directory, and it is this directory that is copied when you cloned the Team Echo repository from Github.

The working directory, is a single checkout at one point in time. When you checkout a specific version of your project, the files are decompressed from the .git directory and extracted into the root project directory, allowing the developer to use and modify the files.

The staging area is an intermediate holding area, internally represented as a file, which contains information about what changes will go into your next commit. The staging area allows one to queue more than one change to go into a commit, which allows developers to cluster groups of changes together. Some literature refers to the staging area as the index.

The git workflow works as follow:

1. You create, modify and delete files in your working directory
2. The changed files, which included files which have been created and deleted, are then added to the staging area. This means snapshots are added to the staging area.
3. The staging area is then commited, which takes the snapshots of the files as they are in the staging area, and adds it into the git database.
4. Wash and repeat

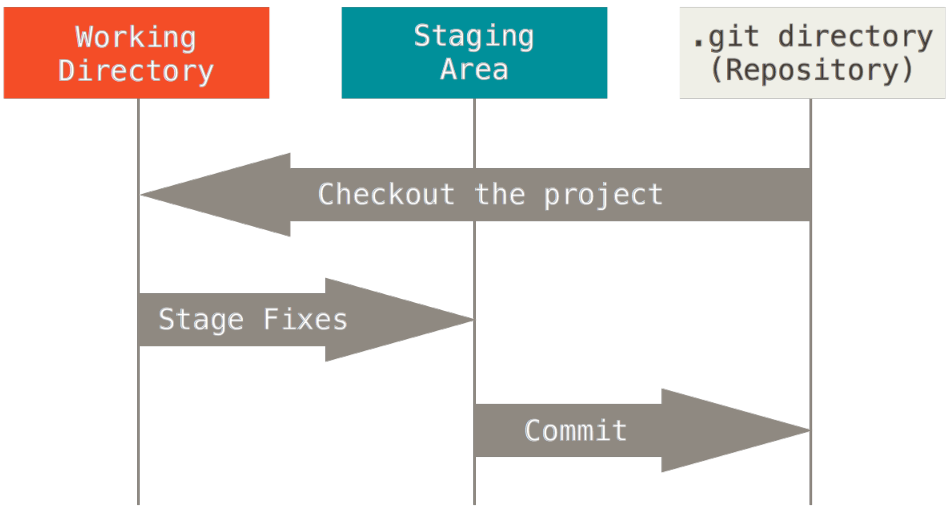

Figure 1: Working directory, staging area, and Git directory. (Source: `https://github.com/progit/progit2/blob/master/book/01-introduction/images/areas.png`)

Your index.html file is now in the staging area. Lets finish off by adding this file into the git database, by using the command

```
# git commit −m "Creates a new empty HTML file".
```

Note that git allows one to attach a message to a commit, which allows other developers and yourself in the future to understand what changes a certain commit has made to your source code. As part of self documenting code, communication and understanding by other developers what you did, it is very important to write clear and concise messages of what changes a commit introduces to the source code.

The power of git will now be unleashed. Branches in git allows one to work seperatly from production ready code, allowing one to experiment, add new features, correct bugs etc. Create a new branch called headBranch

```
# git branch headBranch
```

Git has now created a new branch, called headBranch, lets go ahead, and switch to this branch now by using the command,

```
# git checkout headBranch
```

We can ensure that we are on the correct branch by using the command

```
# git status
```

From the output below you will notice that git indicates **On branch headBranch**.

```
On branch headBranch
nothing to commit, working directory clean
```

Go ahead, open the index.html file, and add a html head tag, i.e. you file will now look like this

```
<html>
<head>

</head>
</html>
```

We will now add this file to the staging area, and commit it. Issue the following commands

```
# git add index.html
# git commit -m "Adds head tag to file"
```

We will now checkout the master branch, by using the command

```
# git checkout master
```

Go ahead and open your index.html file. Shock ... ..., there is no head tag. I know, quite shocking, but it is correct. This is one of the most powerful features of git, when you want to experiment on your own code, test some stuff, without breaking it, you can always create a new branch to test your crazy experiments, and throw the branch away if it doesn't work. However, should your code work, you can always add your new changes to your old code, but we are jumping ahead.

We will now checkout a new branch, called bodyBranch and add the body tags to the index.html file. You can repeat the same procedure as was done for the headBranch, just replaceing the word head with body. However, if you are unsure here are the commands

```
# git branch bodyBranch
# git checkout bodyBranch
```

Before you commit your code, make sure that your index.html file looks like this:

```
<html>
<body>

</body>
</html>
```

```
# git add index.html
# git commit -m "Adds body tag to file"
```

After you have commited your code, we will switch back to the master branch again, using

```
# git checkout master
```

We will now update our file on the master branch, add a DOCTYPE declaration to the file. Ensure your index.html file looks like this, before commiting.

```
<!DOCTYPE html>
<html>

</html>
```

Add this file to the staging area and commit it with a message of "Added DOCTYPE declaration".

```
# git add index.html
# git commit -m "Added DOCTYPE declaration"
```

We will now use the power of git to combine all your hardwork to create a complete html document. Important to note, that when one merges git branches, you always merge the branch you give as a terminal paramter into the current active branch, given by git status command. Hence we want to import our work from headBranch into the master branch, hence we need to ensure that we are on the master branch. This can be checked as before by typing

```
# git status
```

or using the command

```
# git branch
```

and ensuring that the master branch has an asterisk on the left hand side of it's name. This indicates that it is the current active branch.

To merge the headBranch into the master branch issue the command

```
# git merge headBranch
```

Go ahead and open the index.html file. You should notice that you have the following

```
<!DOCTYPE html>
<html>
<head>

</head>
</html>
```

As you can see, git was smart enough to have understood how it should combine your files. We will now merge the bodyBranch into the master branch, by using the command

```
# git merge bodyBranch
```

You will notice that git indicates that it has encounted a conflict

```
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

This means that git is not sure how it should combine the file, and hence is not going to proceed, but rather wait for you to tell it how to proceed. If you open your index.html file, you will notice git combinded the files, but is showing you which part of the file came from which branch.

```
<!DOCTYPE html>
<html>
<<<<<<< HEAD
<head>

</head>
=======
<body>

</body>
>>>>>>> bodyBranch
</html>
```

It is now up to you as the programmer to indicate to git which parts are correct and how you want the file combined, by editing the file by hand to correct it. In this example, we want all the code, so we simply remove meta data lines such that your file looks like the one given below:

```
<!DOCTYPE html>
<html>
<head>

</head>
<body>

</body>
</html>
```

We will now add this file to the staging area and commit it, using a message such as "Merges bodyBranch into master branch document".

```
# git add index.html
# git commit −m "Merges bodyBranch into master branch document"
```

Now, how will the lecturers see that you are a hard worker or social loffer? Easy type

```
# git log
```

To see the changes you made, you can use the following command

```
# git log −p
```

There is all the history of what we have been doing so far, as well as who committed code and at what time the code was checked in.

1.5 Social side of code sharing

Head on over to Github.com and create a new repository, called cos301tutorial. After creation you will find a similar line to this,

```
git remote add origin git@github.com:andrewbroekman/cos301tutorial.git
```

Copy and paste this line into a terminal. This line will link your repository to the online repository on GitHub. However, we need to upload our git repository to Github, similar to how we upload HTML,CSS and JS files to a webserver using FTP. To upload the files to GitHub, we issue the following command

```
# git push −u origin master
```

This pushes your master branch code, to the origin server. The origin server is GitHub, as can be seen in the code line below.

```
git remote add origin git@github.com:andrewbroekman/cos301tutorial.git
```

The word origin is just a label which is used as an index to specify the URL of the remote server. Advanced git usuage allows push code to multiple servers.

You can now go ahead, and explore your repo on GitHub, ensuring you get familiar with the GitHub interface.

Go ahead, and checkout the headBranch again. We will now push this branch also to Github so that your friend can assist you in writing the head section with you.

```
# git push −u origin headBranch
```

Go explore your headBranch history on Github.

We can now go ahead and a title tag to the head tag. Also, we create a new file, called README, where you can write all about your new site that you will launch. Ensure your index.html looks similar to this

```
<!DOCTYPE html>
<html>
<head>
        <title>COS301 Git Intro</title>
</head>
<body>
```

```

</body>
</html>
```

My README file looks like this

```
That's one small step for [a] man, one giant leap for mankind — Neil Armstrong
```

We will now commit multiple files in one go, issuing the following commands

```
# git add index.html
# git add README
# git commit −m "Adds title tag and README file"
```

Important to note, committing only adds a file to the local repository history, not to Github. You can verify this, by checking the commits in the headBranch to verify that this latest commit was not added to Github. Why, we haven't upload our changes yet, so go ahead and upload your changes.

```
# git push −u origin headBranch
```

Your changes will now be visible in Github, verify this.

Finally, checkout your master branch or bodyBranch, and see if they contain a README.md file?

1.6 Why do I need to use branches?

As noted in class, your master branch should always contain production ready code, i.e. this branch should be able to be checked out anytime of the day or night and be able to compile without errors, and the system should be in a fully working state. To achieve this, we use git branches, to accomplish this, however branches introduce a lot of administration, which we have not done properly in the above tutorial, that should be done before a branch is merged. Hence based on the following blog post by Vincent Driessen titled "A successful Git branching model" located at `http://nvie.com/posts/a-successful-git-branching-model/` spurred the open source community to create a branch management tool for git called called git flow.

To install git flow on Ubuntu

```
# sudo apt−get install gitflow
```

In your cos301tutorial, issue the following command

```
# git flow init
```

The above command initiates the git flow tool for this repository. You can go ahead and accept the defaault settings. Refer to this handy cheatsheet `http://danielkummer.github.io/git-flow-cheatsheet/` The nice thing about gitflow, is that it merges the branches in the correct order for you, as well as clean old branches etc. Most git clients today support git flow, as it is widely used in industry.

For a very good tutorial on git flow is presented by GitHub. More information is available at `https://guides.github.com/introduction/flow/`

1.7 Final comments on the use of GIT

Now that you have been introduced to most aspects of git, it is up to you to go and learn and get comfortable with git, as this is vital for your code production quality and speed. A few notes before you start working in a full on project. Before your work do the following:

Git pull on master branch to update your code:

```
# git pull
```

Now you can start working on your code (On a different branch)

When you have finished with your code add it to the staging area:

```
# git add <file>
# git add <file>
```

Now you commit your work to your local repository:

```
# git commit -m "Commit Message"
```

Finally push your work to a branch

```
# git push -u origin branchName
```

1.8 COS301 Mini Project Assignment

Once you are comfortable with git, go ahead and add your details to the CONTRIBUTORS.md file, commit and push your changes to the echo repository.

For more information about git and tutorials, refer to the following links:

- `https://git-scm.com/`
- `http://book.git-scm.com/`
- `https://www.atlassian.com/git/tutorials/`
- `http://gitimmersion.com/`

2 LaTeX

2.1 What is LaTeX?

LaTeX is a document preparation software and document markup language. LaTeX is based on Donald Knuth's TeX typesetting software, and LaTeX itself is written in the TeX macro language. LaTeX differs from traditional word processors, which are based on the WYSIWIG (What You See Is What You Get) principle, rather letting the author focus on the content of the article in question, and leaving styling up to the TeX system. The author can however provide hints and markup certain content to provide the system with more information about the required layout and appearance.

2.2 Installing LaTeX on Ubuntu

To install LaTeX on Ubuntu, issue the following commands

```
sudo apt-get update
sudo apt-get install texlive-full
```

2.3 Installing LaTeXon Windows

Microsoft Window users can install LaTeX by using the MiKTeX package. More information is available at `http://miktex.org/`.

2.4 LaTeX document

To see how a LaTeX document looks like, navigate to the Team Echo repository, open the docs folder, whereafter open the intro folder. Next open the tutorial.tex file in your favorite text editor to see how the document looks.

To compile this document, issue the following command

```
# pdflatex tutorial.tex
```

This will produce a PDF file called, tutorial.pdf.

For more information about LaTeX, refer to the following links:

- `https://en.wikibooks.org/wiki/LaTeX/Introduction`

- `https://latex-project.org/intro.html`
- `http://www.andy-roberts.net/writing/latex`
- `https://tobi.oetiker.ch/lshort/lshort.pdf`

Also refer to the support materials section, specifically section 3.2.

3 Support Materials

3.1 JetBrains IntelliJ IDEA Ultimate

The IntelliJ IDEA IDE from JetBrains is of the most advanced and inteliigent Java IDE's avaiable. However with the time and resources invested by JetBrains, it is also one of the most expensive IDE's on the market, especially to students, at a cost of US$499.00. However as students of the University of Pretoria, we are however granted the privilege of obtaining this software for free.

The reason I recommend this software, is that it has very good support for DI (Dependency Injection) framworks and unit testing frameworks.

To acquire this software, head on over to `https://www.jetbrains.com/shop/eform/students`, complete the form and enjoy your new software. To ensure you will be able to get access to the software, ensure in the **Apply with** section you select **Univesity email address**. In the email field, use your UP GMail account to register, i.e. **uXXXXXXXX@tuks.co.za**, where uXXXXXXXX is your student number.

3.2 Safari Tech Books

A lot of UP students are not aware that they have free and legal access to over 40 514 technology books. In order to access Safari Tech Books

1. Naviagte to `http://www.ais.up.ac.za/`
2. Move your cursor over the **E-Resources** tab, allowing the drop down menu to open.
3. Select **E-Book Collections** - Direct Link `http://www.library.up.ac.za/ebooks/collmain.htm`
4. Select **Safari Tech Books Online**
5. Enter your student number, without the "u" and library password
6. Enjoy your access to free, legal tech books.

For more information on LaTeX, do a search on LaTeX. You will notice this site contains books on all technology topics, from server administrations such as how to setup BIND (DNS server), Java Game development, LaTeX, PHP, Android dev, Java EE, Spring, Scale etc.