INFO1111: Computing 1A Professionalism

2023 Semester 1

Skills: Team Project Report

Submission number: 1

Github link: https://github.sydney.edu.au/mbli9416/INFO1111_CC15_05

Team Members:

Name	Student ID	Levels already achieved	Levels being attempted	Selected Major
BLIGHT, Matthew	490423105	A, B	C	Computer Science
LOUIS, Tobit	530487713	X	A	Data Science
TRAN, Jordan	530503851	A	B, C	SW Development
DANG, Chloe	530476094	X	A	Cyber Security

Contents

1. Teamwork

1.1. Developing industry skills

To be trained/assisted by people who are well versed in new areas you are interested in proves to be an invaluable means of enriching your learning, given explanations are tailored to the individual and far more engaging than online courses. These people are inclusive of your superiors, coworkers or other members of your network. This approach is most appropriate once a basic understanding of the topic has been attained and you wish to master it whilst challenging yourself by developing complex projects and solving advanced problems. In this way, help is individualised and readily available. For instance, taking initiative to request and negotiate with a co-worker experienced in the field of interest for assistance - not only do they know real world applications, but also offer extensive knowledge to deepen your own understanding.

Online study courses are most appropriate for when someone wants to gain a comprehensive understanding of the programming language or topic in general. They offer the learner the opportunity to develop employable skills through the convenience of an online interface and even gain qualifications. They do however require an extensive amount of time and dedication to complete and therefore, may not be suitable for beginner programmers looking to extract quick and easy solutions as the information in these courses is often embedded amongst modules, or behind other boundaries such as having to complete previous tasks. Thus the online study courses are suitable for those willing to undertake a complete overview of a program.

Practical coding by trial-and-error, in conjunction with google (i.e. stack overflow), is another effective approach that all programmers despite their respective field or level should use for their continuous learning. Coding by trial-and-error is absolutely necessary if a programmer strives to develop new skills and proficiency in arising programming languages. This approach enables a more advanced level of skill honing, problem-solving and analysis skills to debug errors successfully (via research and google) and develop logically complex programs. Learning by trial-and-error serves as an effective method to ease into and understand all the nuanced syntaxes of a programming language. However, the method usually takes an extensive amount of time to work at before a programmer could see success for employability qualifications. Thus, this approach in conjunction with google / stack overflow is an essential part of continuous learning for programmers to hone their skills within their respective fields.

Watching educational YouTube videos is a useful approach for learning a new tool or language. Coding tutorials and other similar videos give a great foundation for understanding the workflow of the program. This can be a helpful place to start when the tool is completely foreign, as you get to see just what it looks like to use the tool in an effective way. The downside to this approach is that learners can get stuck just watching people use the tool, rather than practically using the tool themselves. The best way to gain proficiency in a skill is to acquire experience doing it. Therefore, if a programmer is neglecting to combine practical applications with the YouTube tutorials they watch, they will fail to truly make any progress in their learning and proficiency.

A final approach that can be brought to continual learning is reading the official documentation for whichever tool is being learned. While documentation can be an effective tool as a reference to understand the framework of a program, it is not very practical for learning new languages when used on its own. Programming documentation often goes into much more detail than is necessary for a basic understanding, and can be hard to decipher due to the jargon being used. It is difficult to master reading on its own, hence it is best used alongside other resources or when encountering problems in implementation.

Overall, reading programming documentation on its own is not a very effective approach to learning a programming language or tool, but can be useful if a learner gets stuck on their journey.

1.2. Submission 1 contribution overview

After initially receiving the group project, the team discussed various strategies to evenly split the tasks presented within the Teamwork section 1.1. We concluded it was best to brainstorm five of the approaches to continual learning together, and then assign each member to complete one. Given our team comprises four members, we took initiative to complete the remaining approach during a group meeting. As for individual level A attempts, all members were able to complete their section and pull/commit changes successfully. Within the final group meeting, we finalised and ensured the TEX file contained all necessary changes. Overall, there were equal contributions and a productive work flow between members.

1.3. Submission 2 contribution overview

For our second submission, we started by discussing the feedback from our first submission and applying the relevant changes. Each of us made our own individual contributions through GitHub in our own time. Overall there was an equal contribution between members.

1.4. Submission 3 contribution overview

As above, but completed for submission 3

2. Level A: Basic Skills

Level A focuses on basic technical skills (related to LaTeX and Git) and the types of skills used in different computing jobs. Each member of the team should individually complete their subsection below. You should begin by allocating to each team member a different major to focus on (i.e. one of: Computer Science; Data Science; Software Development; Cyber Security). *If you have a fifth member, then your tutor will suggest a fifth topic to cover.* This allocation should be specified above (see lines 36-55 in the LaTeX file).

You should then begin by looking at the list of skills identified within SFIA (Skills Framework for the Information Age) [1]. Then select two skills from the complete list:

1. The skill in which you believe you are currently the strongest and which is relevant to the major you have selected. You should explain why this skill is necessary for that major, and what evidence you currently have that demonstrates your strength in this area. (Target = 200-400 words).

2. The skill in which you believe you are currently the weakest but which is important to the major you have selected. You should explain why this skill is necessary for that major, and what you can do to improve your capability in this area. (Target = 200-400 words).

You will need to integrate your information into the shared collaborative LaTeX document and compile the result.

2.1. Skills for Computer Science: BLIGHT, Matthew

Strongest Skill: Emerging Technology Monitoring [1]

The ability to regularly monitor for new and emerging technologies is an invaluable skill within the field of computer science. Staying on top of new tools and techniques can help computer scientists solve problems and work more efficiently. For example, a newly developed programming language could help developers create new software at a quicker pace, or with fewer bugs.

Maintaining pace with the languages and frameworks of today can also assist in effective collaboration and communication between peers. It is much easier for computer scientists to create new software and share meaningful ideas about the future with their colleagues when they are all on the same page about the newest developments in the field.

Moreover, having an up-to-date knowledge on current technological advancements could help computer scientists predict future trends in the industry. Today's digital landscape is rapidly changing, with evermore products, programming languages and methodologies being developed everyday. As these new technologies become accessible, it is vital that computer scientists keep their fingers on the pulse so that they can maintain a competitive advantage in their field.

I have chosen this skill as my strongest skill that relates to the field of Computer Science, as I have a keen interest in being aware of recent developments in technology and research. I love to read online articles and consume many other resources in order to understand the newest technologies and changes in the industry. For example, when as soon as OpenAI publicly released ChatGPT, I was incredibly interested in how this technology worked, and spent time researching the implications that it and other generative AI models would have for the industry and society at large. While I do feel that I have a strength in this skill, there are still many advancements I could make such as

subscribing to a daily/weekly email outlining the newest technological developments in particular fields.

Weakest Skill: Scientific Modelling [1]

Scientific modelling is an important skill as it allows researchers to analyse complex systems and predict their behaviour under different conditions. In computer science, scientific modelling is used to simulate and study the performance of algorithms, software systems, and computer networks. This skill is essential for researchers to develop new technologies, optimise existing systems, and solve real-world problems [2].

Since the field of computer science intersects a wide array of disciplines, the applications of scientific modelling are extremely varied. Computational models are relied on far and wide; from the simulation and analysis of developmental biology in plants [3], to the study of human collective behaviour in the field of cognitive science [4]. As these models only become more powerful and efficient, their applications will exponentially increase. Hence, scientific modelling will continue to be an invaluable and highly sought-after skill for computer scientists.

The best approach that I can bring to improving my capability in scientific modelling is to take the courses in statistics, mathematics, and machine learning provided by my degree. These courses provide the foundational knowledge and skills necessary for building scientific models. Once this foundation is strong, I can expose myself to new techniques and methodologies for scientific modelling by participating in research projects and attending conferences/events that dive deeper into the concepts and tools used in scientific modelling.

Furthermore, a proficiency in relevant programming languages such as Python and MATLAB is essential for building scientific models. To enhance my skills in these areas, I can take online courses and begin creating small projects to start my journey.

2.2. Skills for Data Science: LOUIS, Tobit

Strongest Skill

The SFIA skill I have identified as being strongest is research. It is highly relevant to the Data Science major because of the following: Data scientists need to research in order to understand characteristics from data and gain meaningful insights and interpretations. Research skills are essential for data science projects that involve working with emerging technologies such as artificial intelligence, big data, and machine learning. As a first-year data science student, one of the most important skills to develop is the ability to conduct research. The computing industry is constantly evolving, and data scientists need to stay current with the latest technologies, methods, and algorithms.

A strong research skillset will enable a data scientist to understand and interpret complex datasets, develop new algorithms or models, and identify potential areas for improvement in existing models [5]. Research skills are also essential for data science projects that involve emerging technologies, such as artificial intelligence and machine learning.

Moreover, the research process involves a series of iterative steps, including problem definition, data collection and analysis, hypothesis formulation, testing and validation, and finally, communication of results [6]. A data scientist with strong research skills will be able to navigate these steps with ease and produce actionable insights that drive business value [5]. For a first-year data science student, it is critical to have a strong research base and be able to conduct thorough research, stay current with the latest trends and advancements, and apply data-driven solutions to identified areas.

Throughout my schooling, I have developed the sound research skills needed to formulate essays, reports and other analysis pieces and this is translatable to the data science field.

Weakest Skill

The skill I believe I am weakest in is numerical analysis. This is largely due to the foundation of this study being extremely theoretical and only implementable after a comprehensive understanding which makes it difficult to engage in initially. Numerical analysis is a key component of data science as it plays a crucial role in analysing data. It involves the development of mathematical models and algorithms to analyse large data sets [7].

A key reason for the importance of numerical analysis in this field is that it provides a factual foundation for developing data-driven solutions. Quantitative data often gives the interpreter a concise and straightforward outcome and is not generally subjective. This allows data scientists to make informed decisions that impact a wide range of industries, including finance, business, healthcare and technology. It also helps scientists identify trends and patterns within the data, to extract relevant solutions.

Numerical analysis is also essential for creating models and algorithms that may be applied to improve data processing and analysis. These algorithms can aid in reducing the time and resources needed for data analysis, enabling quicker and more effective insight generation. This is crucial in fields like finance and healthcare where decisions with a tight timeline must be based on data analysis [8]. The capacity of numerical analysis to offer reliable statistical estimates of uncertainty is a key component of data science. This is significant since a lot of data-driven decisions rely on projections based on faulty or incomplete data. Data scientists can take this uncertainty into account and create more robust and dependable models by using numerical analysis.

I believe I can improve my capability in this area by further understanding the theoretical information, in relation to its practical implementation. This will help me be able to analyse data to a higher level and implement real solutions.

2.3. Skills for SW Development: TRAN, Jordan

1. The skill that I believe I am strongest in and is also majorly relevant to the software development field is testing. Testing refers to investigating products, systems and services to assess whether the specified or unspecified requirements and characteristics are met. It involves the use of manual testing or automated testing where test cases are created defining the test conditions for a given requirement. Moreover, test outcomes must also be recorded and analysed to validate the program and improve upon errors. The skill in testing is fundamental to the nature of software development involving the process of planning, designing ,creating, testing and supporting software products. Software development as a whole works to conceive a successful product that guarantees quality assurance in which testing is essential for meeting these standards. Thus, expertise in testing is required to determine whether the output of a software product meets the benchmark and assure that the product is up to standard for consumers. I believe that I am most well-versed in testing given my experience in school to create software projects via an agile approach requiring methods of testing such as black box testing where I had designed a test case to isolate a specific function comparing the expected to actual output. Moreover, I had undergone work experience that required testing for products in development using an array of tools and testing approaches to further analyse and suggest improvements upon a more quality assured product.

2. Application support is another communicational skill that is fundamental to the software development field and a skill that I believe to be the weakest in, given my incon-

sistency in providing information and unorganised work-life balance to routinely schedule for maintenance of products. Application support involves the deliverance of management, technical and administrative services to ensure a successful live application. Application support is a necessity in the software development cycle as a multitude of factors for a high-quality and reliable product need continual assistance post the development stage in order to meet a successful live application. For example, guidance and training for the users following new software releases, monitoring application performance, updating documentation and capturing user feedback are all application support skills that are a must-needed skill for every software developer in order to successfully produce a reliable product. Thus, Application support is by far one of the most important skills needed in order to become a successful software developer. However, I believe this to be one of my weakest skills due to the lack of experience in handling support post development as I have no experience in developing a live application (i.e. cloud-based) thus, cannot deem myself sufficient enough to deliver management, technical or administrative services for any customers. The most appropriate way to improve my capabilities in this area is to look for software development internships in my 3rd or 4th year and learn how to routinely schedule my assistance to customers for live-applications that have already been developed in these companies.

2.4. Skills for Cyber Security: DANG, Chloe

Strongest skill.
I believe the skill I am currently most proficient at is teaching. Teaching is defined as the delivery and assessment of syllabus within an orderly environment for educational purposes, wherein instructors help foster a comprehensive understanding of principles and practices in regards to a specific topic [1]. As denoted by SFIA [1] in the context of computing and information technology, areas addressed typically include:

- Regular digital skills utilised to advantage and contribute to daily life and work in terms of the digital world.
- Extending one's understanding of particular topics eg. recently emerging technologies and new applications for current technologies.
- The notion of computational thinking (ie. the thought process of systematically formulating problems and representing their solution as simple, broken-down steps) and the application of computational concepts to daily life and work.

Teaching is an especially critical skill in cybersecurity in terms of studying recently emerging technologies and new applications - keeping up to date with potential ways adversaries may attack a system or network is vital to productive preparation and successful defence for an organisation. To teach thoroughly and efficiently is key to maintaining an advantage and powerful digital fortification over security hackers, as equipping lesser experienced employees with adequate skill reinforces security by combatting exploits and mitigating newly posed risks. To expand upon why I believe teaching is a strength of mine, I have experience tutoring high school students in a formal classroom setting as a part time job. In doing so, I have effectively learnt to introduce and explain concepts in a coherent, concise manner as well as be proactive in ensuring the understanding of others. Thus, these skills may be applied in the context of priming others to react and manage cyberattacks in a precise, orderly manner.

Weakest skill.
The skill I am weakest at which is also most relevant to cybersecurity is penetration testing

- given I am inexperienced in the technology industry in all aspects. Penetration testing is defined as examining the effectiveness of security controls (ie. parameters installed to detect, avoid and prevent cyberattacks) by reproducing techniques utilised by potential hackers. According to SFIA [1] it involves:

- The safe exploitation of security faults.
- Investigating methods used by adversaries to disrupt security goals or achieve particular objectives.
- Evaluating how effective defences/mitigation controls (either current or future) withstand cyberattacks.
- Reinforcing the security of networks, systems, and applications.
- The identification of different vulnerabilities and risks posed to the business.
- The assessment of network, infrastructure, web and mobile applications.
- Checking patch levels and configurations.
- Social engineering ie. the use of psychological manipulation to gain control over a system or network.

Penetration testing is crucial to the validation of security within an organisation's systems and networks, as well as in assisting organisations to design effective, functional security controls and better security processes. As aforementioned, I deem it my weakest skill due to inexperience. In order to improve my capability in this area, I must first gain an adequate understanding of the fundamental basics related to cybersecurity and extend further (this may be done through selecting cybersecurity as my major during my 2nd year of my advanced computing course). There, I will grasp an extensive comprehension of how to perform penetration testing alongside other concepts, and eventually be able to apply it in industrial settings once I gain an internship or official position in an organisation.

3. Level B: Tools

Level B focuses on exploration of key tools used within professional computing employment. All companies make use of a range of technologies and tools (often as part of a tech stack). These tools might be implementation languages; design tools; data analysis tools; collaboration technologies, etc. Each student should identify two tools that are widely used in industry and which relate to the major you are focusing on for this project. You should then describe:

1. The main functionality of those tools;
2. The ways in which those tools are used;
3. Any weaknesses or limitations of those tools.

As examples (which you shouldn't now use): Computer Science: eclipse; Software Development: github; Cyber Security: Wireshark; Data Science: Hadoop.

Note also that no two students in the same tutorial should choose the same tools, so your tutor will maintain a list of those that have already been selected. You should therefore check this list and then confirm your choice with your tutor prior to researching your proposed tools and spending time writing about them. (Target = ∼200-400 words per tool).

Also, in order to achieve level B each student needs to be able to demonstrate capability with git and compilation of LaTeX documents from the command line. To demonstrate this, your team (or at least those members who are aiming to attempt level B) should do the following:

1. Select one member to:
   (a) Create a local github repository for the project. This repository should contain the main LaTeX documents, as well as a subdirectory called "screengrabs";
   (b) Create a repository on github for the project;
   (c) Connect your local repository to the remote github repo;
   (d) Push your local repository contents to the remote repo;
   (e) Add all team members (and your tutor and unit coordinator) as members to the remote repo;
2. Each additional group member should then clone the remote repo;
3. Each member aiming to achieve level B should then be able to use the remote repo (and pushing and pulling changes) to demonstrate collaborative editing of the LaTeX documents.
4. And each member aiming to achieve level B should also do a screengrab (or multiple screengrabs) showing their local successful compilation, on the command line, of the final LaTeX document. This should be added to the screengrabs folder in your local repo and then pushed to the remote repo so that your tutor can view it.

3.1. Tools for Computer Science: BLIGHT, Matthew

Tool 1: Vim

Vim, short for "Vi Improved", is a powerful open-sourced text editor created by Bram Moolenaar in 1991. It is an updated version of the original vi editor that was created by

Bill Joy in the 1970s for Unix operating systems [9]. Vim has become an essential tool for computer science due to its incredible efficiency and versatility.

The primary function of Vim is to create and edit text files through a graphical interface in the terminal. Being a mouse-less text editor, its users can perform tasks quickly without the need to lift their fingers off the keyboard, allowing programmers to create and edit code with great efficiency and speed. Some of Vim's main functionality includes the ability to move around a document quickly, search for specific text patterns, and perform complex text operations with ease. Moreover, Vim is highly configurable with a multitude of plugins available online for any kind of customisations. This allows users to adapt it to their preferred work style and increase productivity.

Vim is widely used in computer science by programmers who spend a lot of time writing, editing and testing code. It is most commonly encountered from the command line as it is naturally installed on most Unix systems, however it can also be used in conjunction with other tools. For example, Vim can be downloaded as a plugin for most integrated development environments (IDEs) such as VSCode or Eclipse. This allows the efficiency of Vim's shortcuts and modality to be integrated with whatever debugging tools and other powerful engines available on your preferred IDE.

Despite its many strengths, Vim has some weaknesses and limitations. One of the most significant limitations is its steep learning curve, which can make it challenging for new users to get started. Vim requires a significant investment of time to learn its various commands and modes, which can be frustrating for some users. Additionally, Vim is primarily a text editor and, on its own, does not offer the same level of functionality as an IDE for larger programming projects.

In conclusion, Vim is a powerful and versatile tool that is widely used in computer science. Its speed, efficiency, and customisation options make it an ideal text editor for developers who spend a lot of time writing code. Although Vim has a steep learning curve and some limitations, its many strengths make it an essential tool for computer science professionals.

Tool 2: TensorFlow

TensorFlow is an open-source software library that uses data flow graphs for machine learning. Developed by the Google Brain team, TensorFlow has become a popular tool in the field of computer science due to its flexibility and ease of use [10]. The library enables developers to build and train machine learning models, and execute complex numerical computations efficiently.

One of the main functionalities of TensorFlow is its ability to build and train neural networks, which are fed large amounts of data to make predictions and classifications in various applications [11]. The TensorFlow framework inputs data in the form of multidimensional arrays called 'tensors', and builds a computational graph defining the data flow between the various mathematical operations, called 'nodes' [12]. This architecture can be used to create and train various types of neural networks, allowing developers to create models that can perform tasks such as image recognition, speech recognition, and natural language processing.

One of the main advantages of using TensorFlow is its scalability. The library is designed to work with both CPUs and GPUs, allowing developers to train and run models efficiently on large datasets. TensorFlow also provides distributed computing capabilities, enabling developers to run computations across multiple machines.

While TensorFlow has many advantages, it does have some limitations. The library can be difficult to set up and use for beginners, and some of its features require knowledge

of advanced computational and mathematical concepts. Furthermore, TensorFlow can be resource-intensive, requiring significant computational power and memory to train complex models.

Overall, TensorFlow is a powerful tool for computer science that enables developers to build and train complex machine learning models. With its scalability, versatility, and range of features, TensorFlow has become an essential tool for researchers and developers working in the field of artificial intelligence. However, its complexity and resource requirements may limit its accessibility for some users.

3.2. Tools for Data Science: LOUIS, Tobit

Your text goes here

3.3. Tools for SW Development: TRAN, Jordan

Tool 1 : Docker

Docker is a powerful platform that provides a standardized way to package, distribute, and run applications. Its main functionality revolves around containerization, which allows applications to be bundled with their dependencies into isolated environments called containers. [13] This ensures that applications run consistently across different systems, regardless of the underlying infrastructure. Docker simplifies the development and deployment process by abstracting away the complexities of system dependencies. With Docker, developers can package their applications along with all the required libraries, frameworks, and tools into a container. These containers are lightweight, portable, and can be easily deployed on any system that has Docker installed.

[14] The way Docker is used involves creating a Dockerfile, a text file that contains instructions to build a Docker image. The Dockerfile specifies the base image, adds the necessary dependencies, sets configuration options, and defines the commands to run when the container starts. Once the Dockerfile is created, developers use the Docker command-line interface to build the image and run containers based on that image. Docker has gained popularity for various reasons. It enables developers to work in isolated environments, avoiding conflicts between different applications and dependencies. It also facilitates scalability, as containers can be easily replicated and distributed across multiple hosts. Furthermore, Docker supports automation and streamlines the continuous integration and deployment (CI/CD) pipelines.

However, Docker does have some weaknesses and limitations. First, while containers provide isolation, they still share the same underlying operating system kernel. This means that any vulnerability or misconfiguration at the kernel level can affect all containers running on the same host. Additionally, managing stateful applications, such as databases, can be more challenging with Docker, as containers are typically designed to be stateless as it does not provide any meaningful solutions for data backup/recovery [15].

Tool 2 : Lucidcharts

Lucidchart is a design tool that allows users to create and collaborate on types of visualizations and diagrams. It offers a wide range of functionality and features that make it popular among individuals and teams in software development. [16] The main functionality of Lucidchart lies in its ability to enable users to create professional and visually appealing diagrams with ease. It provides an intuitive drag-and-drop interface, allowing users to add and arrange shapes, lines, and text boxes to create different types of diagrams

such as flowcharts, wireframes and UML diagrams. The prebuilt libraries and shapes make the design tool effective in time. Lucidchart offers collaboration through real time editing. Multiple users can work on the same diagram, making it ideal for software development teams that need to design, and document complex programs. The tool also allows users to track changes, leave comments, and receive feedback, promoting effective communication and teamwork. However, there are some limitations to consider when using Lucidchart. While it offers a wide range of diagramming capabilities, it may not have advanced features specifically tailored to software development, such as code generation or integration with development environments. Users who require highly specialized diagramming tools for software engineering tasks may find Lucidchart lacking in certain areas. Additionally, the free version of Lucidchart has some limitations on the number of documents and shapes available, which may be a consideration for users with extensive diagramming needs.

3.4. Tools for Cyber Security: DANG, Chloe

Your text goes here

4. Level C: Advanced Skills

Level C focuses on more advanced technical skills in LaTeX and Git.

The following is a list of advanced Git and LaTeX skills/features. Each student in your team should select a different pair of items from each list (e.g. you might choose "Resetting and Tags" from the git list, and "Cross-referencing and Custom commands" from the LateX list). You then need to demonstrate actual use of each item (either through activity in Git, or through including items in this report). (Target = ∼100-200 words per student for each feature).

- Git
  - Rebasing and Ignoring files
  - Forking and Special files
  - Resetting and Tags
  - Reverting and Automated merges
  - Hooks and Tags
- LaTeX
  - Cross-referencing and Custom commands
  - Footnotes/margin notes and creating new environments
  - Floating figures and editing style sheets
  - Graphics and advanced mathematical equations
  - Macros and hyperlinks

4.1. Advanced skills: BLIGHT, Matthew

Git: Rebasing

`git rebase` is a useful command for integrating changes from one branch into another. While `git merge` is used to solve the same problem, the two commands have important differences. The key difference is that `git merge` creates a new "merge commit" that combines the histories from both branches, whereas `git rebase` re-writes the project history by moving/combining a sequence of commits to a new base commit [17]. For example, consider a repository in which you are developing a new feature on branch `feature`, but there have been new commits in the `master` branch. At this stage, our `master` branch log is below:

```
mattblight@Matthews-MacBook-Air git_rebase % git log
commit 1b48600be18ad5171b7f88cbf02901b5ecee97b5 (HEAD -> master)
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:52:31 2023 +1000

    Fix Bug

commit c2b1574eb451479c940ae1d23dd642969e963c3b
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:46:59 2023 +1000

    Add More Text

commit 4018f8337a14e7e4b3af9ac999e3d13433703847
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:43:22 2023 +1000

    Initial commit
```

And our `feature` branch log is below (note that the 'Add More Text' and 'Fix Bug' commits are not here):

```
mattblight@Matthews-MacBook-Air git_rebase % git log
commit 35551e4fcde8ebe3de1fb8a5b453b790a2abc413 (HEAD -> feature)
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:45:50 2023 +1000

    Debug Cool New Feature

commit cc4ead826dff2a19c2d706d2bfa8735c57d32bf5
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:44:54 2023 +1000

    Create Cool New Feature

commit 4018f8337a14e7e4b3af9ac999e3d13433703847
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:43:22 2023 +1000

    Initial commit
```

You realise that the new commits in `master` are useful for your feature, but you don't want to create a new merge commit in your `feature` branch. Instead, you want it to seem like you were working on `feature` from the latest `master` commit. Here, you would use the command `git rebase master`. This creates a new 'base' for the `feature` branch, re-writing the commit history to show `feature` branching off from the newest `master` commit. The rebased log would look like this:

```
mattblight@Matthews-MacBook-Air git_rebase % git log
commit a1272bb28fd59d29962fc05f7fd9ced70253fafe (HEAD -> feature)
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:45:50 2023 +1000

    Debug Cool New Feature

commit 57a15bbe9023a0e6a3d9218f73e123f2a25b1187
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:44:54 2023 +1000

    Create Cool New Feature

commit 1b48600be18ad5171b7f88cbf02901b5ecee97b5 (master)
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:52:31 2023 +1000

    Fix Bug

commit c2b1574eb451479c940ae1d23dd642969e963c3b
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:46:59 2023 +1000

    Add More Text

commit 4018f8337a14e7e4b3af9ac999e3d13433703847
Author: Matthew Blight <mbli9416@uni.sydney.edu.au>
Date:   Thu May 25 09:43:22 2023 +1000

    Initial commit
```

The `git rebase` command is incredibly useful for keeping commit history clean and linear, rather than having multiple merge commits, which can look especially messy if the `master` branch is very active. The main rule to remember with this command is to never use it on public branches, as re-writing commit history can make collaboration incredibly messy [18].

Git: Ignoring Files [19]

Sometimes a project may have files that you don't want to commit to a repository. These may be:

- compiled code, such as `.pyc` files
- hidden systems files, such as `.DS_Store`
- files generated at runtime, such as `.log`

While this problem could be solved by only staging the files needed, this can be extraneous and results in git showing that there are untracked files:

```
mattblight@Matthews-MacBook-Air git_ignore_files % git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt

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

A better way of solving this problem is to create a file called `.gitignore` at the root of the repository, which stores the names of files and directories that you want git to ignore. For our example, our `.gitignore` file contains `ignore_this_file.exe` and `ignore_this_directory`:

```
mattblight@Matthews-MacBook-Air git_ignore_files % cat .gitignore
ignore_this_file.exe
ignore_this_directory
mattblight@Matthews-MacBook-Air git_ignore_files %
```

These files and directories will be automatically ignored when staging and committing, as seen below:

```
mattblight@Matthews-MacBook-Air git_ignore_files % git add .
mattblight@Matthews-MacBook-Air git_ignore_files % git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   file1.txt
        new file:   file2.txt

mattblight@Matthews-MacBook-Air git_ignore_files % git commit -m "Initial Commit"
[master (root-commit) b3104db] Initial Commit
 3 files changed, 4 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 file1.txt
 create mode 100644 file2.txt
```

LaTeX : Footnotes and Margin Notes [20]

Footnotes are a great way to provide extra information to the reader whilst keeping the main flow of the text clean and concise.[1] Footnotes are easy to implement in LaTeX. Simply use the command `\footnote{text}`, and your text will be automatically be placed at the bottom of the page.[2]

Margin notes are a great way to exchange comments between authors in the editorial process, or otherwise to provide extra information to the reader. To insert a margin note, use the command `\marginpar{margin note}`.

Margin notes are inserted in the outside margin, starting from the line where it is defined.

LaTeX : Creating New Environments

Environments in LaTeX are useful for applying specific effects to sections of a document. There are a number of existing LaTeX environment, however it is useful to know how to create new environments. The basic syntax for defining a new environment is `\newenvironment{<env-name>}[<n-args>][<default>]{<begin-code>}{<end-code>}`, where `<env-name>` is the name of the new environment. `<begin-code>` and `<end-code>` is the LaTeX code to be applied at the beginning and end of the environment. `<n-args>` can be used to define an environment with arguments, and `<default>` can be used to define an environment with an optional argument. As an example, we can create a new environment called `boldbox`, which puts text in a centered box and makes it bold. The environment definition would be:

```
\newenvironment{boldbox}
    {\begin{center}
    \begin{tabular}{|p{\textwidth}|}
    \hline\\
    \bfseries
    }
    {
    \\\\\hline
    \end{tabular}
    \end{center}
    }
```

We can this use this just like any other LaTeX environment:

```
\begin{boldbox}
I want this text to be bold in a box!
\end{boldbox}
```

Which results in:

I want this text to be bold in a box!

[1] This is an example of a footnote.
[2] Footnotes will automatically be numbered sequentially.

4.2. Advanced skills: LOUIS, Tobit

Explain your use of the advanced Git and LaTeX features.

4.3. Advanced skills: TRAN, Jordan

Explain your use of Git Forking and Special files and LaTeX Floating Figures and Styling Sheets

4.4. Advanced skills: DANG, Chloe

Explain your use of the advanced Git and LaTeX features.

The floating figure

5. Level D: Evolution of skills

Level D focuses on understanding how professional practice might evolve in the future. Most students in this unit are likely to be at or near the start of your degree, and so it might be anywhere from 3 to 5 years before you really start working in industry full-time – and the technology and ways in which people use them can change significantly in that time.

Your answer to this section you should address the following (Target = ∼500 words):

1. Describe what you believe will be the biggest change in the next 5 years in the tools or technologies that are being actively used in industry practice (in your selected major);

2. Revisit the SFIA framework [1] from level A, and identify the one skill that you believe will have the biggest increase in terms of importance over the next 5 years. You should justify your choice.

5.1. Evolution of Computer Science: BLIGHT, Matthew

Your text goes here

5.2. Evolution of Data Science: LOUIS, Tobit

Your text goes here

5.3. Evolution of SW Development: TRAN, Jordan

Your text goes here

5.4. Evolution of Cyber Security: DANG, Chloe

Your text goes here

Bibliography

[1] SFIA, "The global skills and competency framework for the digital world," 2022, see https://sfia-online.org/en/sfia-8/all-skills-a-z.

[2] National Institute of Biomedical Imaging and Bioengineering, "Computational modelling," 2020, see https://www.nibib.nih.gov/science-education/science-topics/computational-modeling.

[3] P. Prusinkiewicz and A. Runions, "Computational models of plant development and form," *New Phytologist*, vol. 193, no. 3, pp. 549–569, 2012. [Online]. Available: https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.2011.04009.x

[4] R. L. Goldstone and M. A. Janssen, "Computational models of collective behavior," *Trends in Cognitive Sciences*, vol. 9, no. 9, pp. 424–430, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364661305002147

[5] SFIA, "Research," see https://sfia-online.org/en/sfia-8/skills/research.

[6] B. Violino, "Essential skills and traits of elite data scientists," 2018, see https://www.cio.com/article/228620/the-essential-skills-and-traits-of-an-expert-data-scientist.html.

[7] P. Steyn, "Why is statistics important in data science, machine learning, and analytics," 2022, see https://towardsdatascience.com/why-is-statistics-important-in-data-science-machine-learning-and-analytics-92b4a410f686.

[8] M. Hunter, "The 3 pillars of math you need to know to become an effective data analyst," 2022, see https://towardsdatascience.com/the-3-pillars-of-math-you-need-to-know-to-become-an-effective-data-analyst-9af50106ffa1.

[9] Shubhankar Khare, "Vim: What is it and why to use it?" 2023, see https://www.loginradius.com/blog/engineering/vim-getting-started/.

[10] NVIDIA Corporation, "Tensorflow," 2023, see https://www.nvidia.com/en-us/glossary/data-science/tensorflow/.

[11] IBM, "What are neural networks?" 2023, see https://www.ibm.com/topics/neural-networks.

[12] Simplilearn, "Tensorflow tutorial: Your gateway to building machine learning models," 2023, see https://www.simplilearn.com/tutorials/deep-learning-tutorial/tensorflow.

[13] Serdar Yegulalp, "Why you should use docker and containers," 2023, see https://www.infoworld.com/article/3310941/why-you-should-use-docker-and-containers.html.

[14] docker docs, "Docker overview," ????, see https://docs.docker.com/get-started/overview/#:~:text=Docker%20is%20an%20open%20platform,ways%20you%20manage%20your%20applications.

[15] Danish Jamil, "Docker-an overview/pros and cons," 2022, see https://www.linkedin.com/pulse/docker-an-overviewpros-cons-danish-jamil/.

[16] Lucid Content Team, "8 reasons why lucidchart is the perfect microsoft visio® replacement," ????, see https://www.lucidchart.com/blog/reasons-why-lucidchart-is-the-perfect-microsoft-visio-replacement.

[17] Atlassian, "Merging vs. rebasing," 2023, see https://www.atlassian.com/git/tutorials/merging-vs-rebasing.

[18] ——, "git rebase," 2023, see https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase.

[19] ——, ".gitignore," 2023, see https://www.atlassian.com/git/tutorials/saving-changes/gitignore.

[20] WikiBooks, "Latex/footnotes and margin notes," 2023, see https://en.wikibooks.org/wiki/LaTeX/Footnotes_and_Margin_Notes.