Open Source Workshop: Version Control with Git

February 2022

Overview

1. What is Version Control and why do you need it?
2. What is Git?
3. Git within DNB
4. Your Turn

What is Version Control?

- Tracking and managing changes to software (code).
- Changes are tracked in a special kind of 'database'.
- Make a mistake? You can turn back to a previous version of your code.

So what problems does it solve?

- Who made changes to what file?
- `code_final`, `code_finalv2`, `code_latest`, ...
- Not having to keep outcommented code in your code base for later use.

```
# def calculate_price(number_of_items):
#     return number_of_items * 3
def calculate_price(item_name: str, number_of_items: int) -> dict:
    return {item_name: number_of_items * 3}
```

What is Git?

- A type of version control system: software for tracking changes in files.
- Created in 2005, by Linus Torvalds (Linux!).
- By far the most popular and widely used version control system right now.

Github, Bitbucket, Azure DevOps, Gitlab 😟?

- Important to realize that they have one thing in common: **they all support Git as the underlying version control system**.
- This means that if you are familiar with Git, **it does not matter which provider you are using**. You will be using the same commands.
- Only provide different interfaces and toolsets.

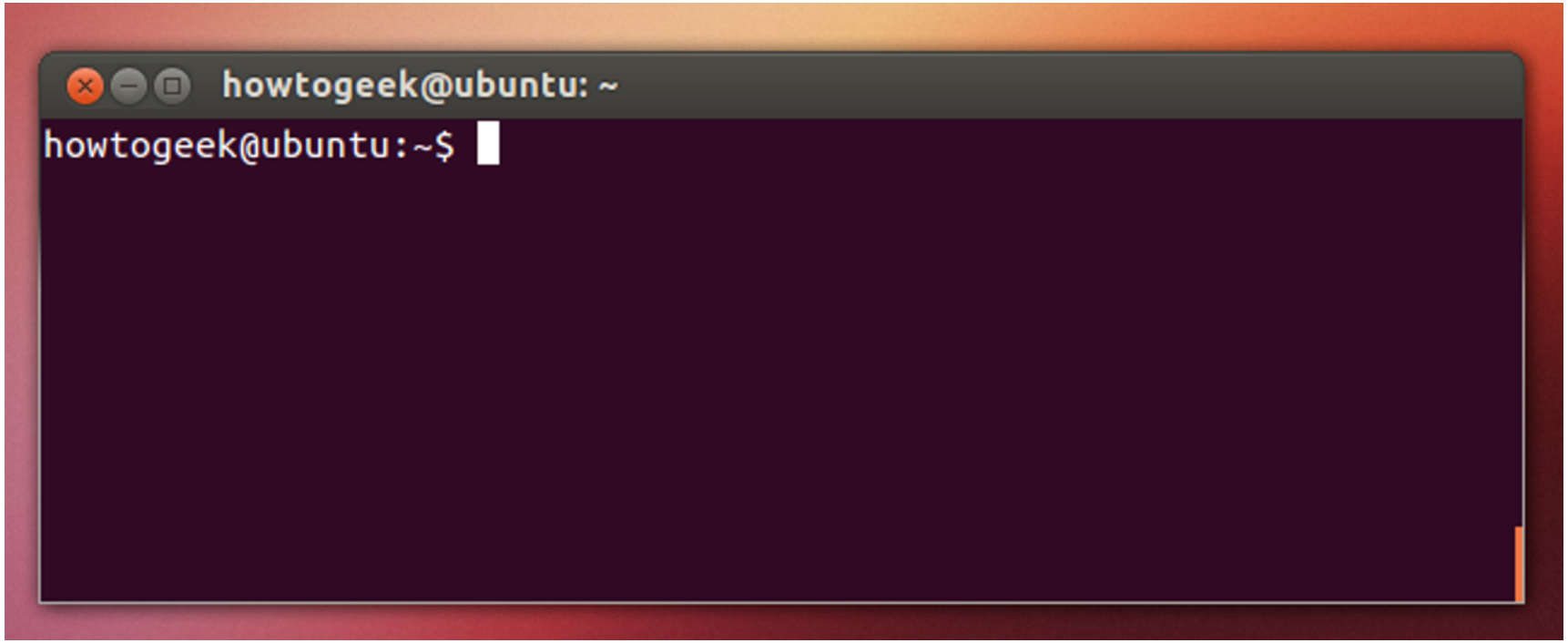
Getting Started with Git

- Git thrives in situations where you are collaborating with others, for example project team members.
- Git is software which you download in order to work with it.
- Git can build a new project directory for you in which it creates a hidden folder where all you additions and changes are stored.
- The resulting snapshots of changes can be used to go back to a version in the past.

Some concepts

- **Repository**: a storage location for software (code)
- When you initialize a folder with Git, Git creates a hidden folder where it, among other things, stores your changes.
- Find a repository on Github you would like to experiment with? **Clone** the repository to a local directory on your computer.
- Once cloned to your computer, **origin** is the name of the remote repository you just cloned.

Git is often used in a CLI

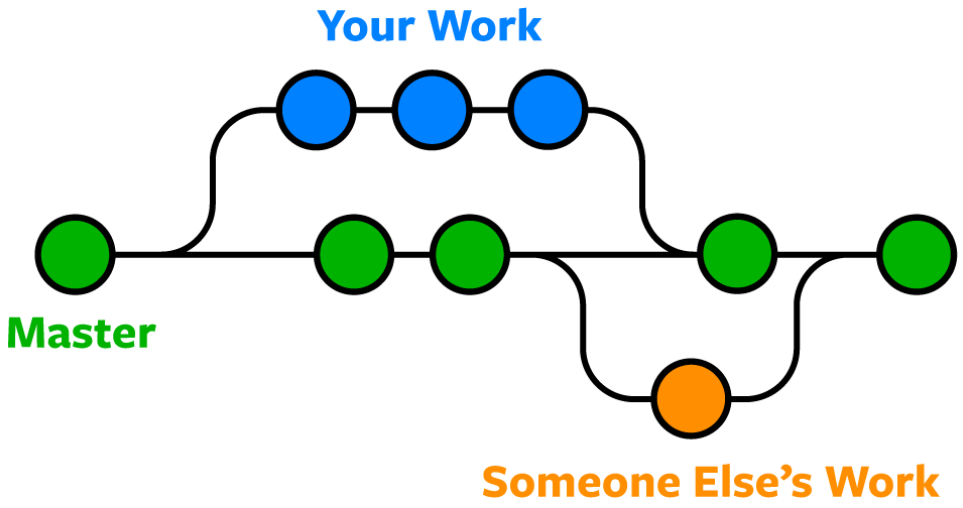


How do I start a Git repository?

- There are multiple ways to start working in a Git repo on your computer:
  - Create a repository inside the version control interface software you are using, e.g. Azure DevOps, Github, etc. Use `git clone` to get that repository to your local computer.
  - Create a folder for your project, open a shell/terminal, type `git init`.
- **Demo**

A Git project

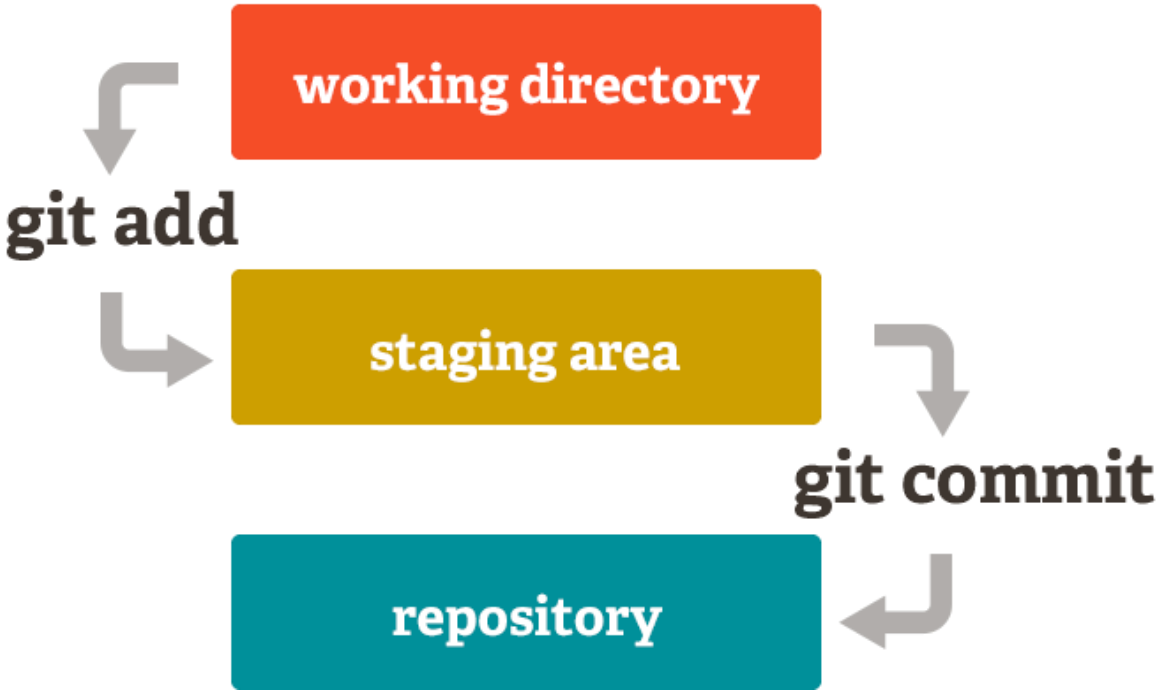
- We now have our project folder with a special hidden `.git` subfolder which will contain all the necessary metadata.
- We are on what you call the `main` **branch**. A **branch** is an independent line of development. The `main` **branch** can be seen as the 'default' branch.
- Note that we have not connected it to any place online; the repository is entirely local.



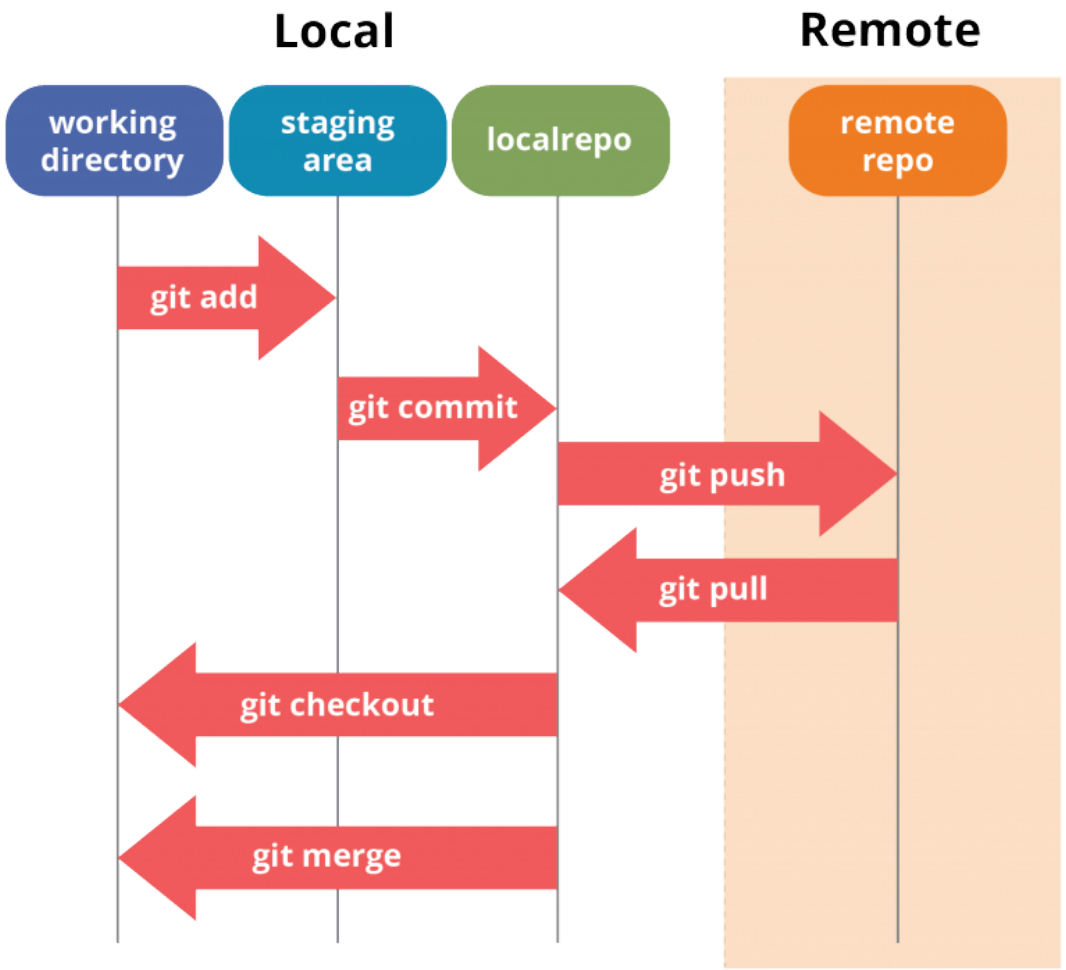
Making your first change

- **add & commit**; use either a terminal or some GUI
  - `git add <filename>`
  - `git commit -m "Commit message describing what you have changed"`
- What this does is add your changes to a staging area, then register the changes to your **local repository**.
- Want to know which files are in which stage and which files are in which area? Use `git status`.
- Finally, **push** your changes to the remote repository: `git push` or `git push <remote> <branch_name>`. Note the remote is often called `origin`.

A bit on Git's inner workings



Visually


Local
Remote
working directory
staging area
localrepo
remote repo
git add
git commit
git push
git pull
git checkout
git merge

Undoing changes

- You can get an overview of your history of commits by using `git log`.
- There are multiple ways, depending on the use case, to undo changes.
- `git revert` is one of them. No history is deleted.

```
git log --oneline
e2f9a78 Revert "Try something crazy"
872fa7e Try something crazy
a1e8fb5 Make some important changes to hello.txt
435b61d Create hello.txt
9773e52 Initial import
```

Undoing changes

- There are multiple ways to undo changes you have made.
- How you undo changes depends on in which stage your change is.
- We won't go into detail on undoing changes, as it might be confusing at first.

Branches

- You might want to try to extend a piece of code with something experimental. In this case, working on the `main` branch is not ideal.
- So, you create a new **branch** to develop a new feature or component:
  `git checkout -b my-feature-branch`
- After you are done, you can merge your changes into the `main` branch, possibly before having had approval to do so (**code review** and **pull request!**), and remove your branch.

💡 Quick summary of what've learned so far

- Git stores the changes you are making in your local repo through a process of **staging** them, then **committing** and **pushing** them to the remote repo. **Pulling**, on the other hand, updates your local repository from the remote repo.
- Git works with **branches**: usually a `main` branch consists of code that is working and ready to be used. You work on code in different branches and **merge** completed code back into the `main` branch.

Let's revisit the `main` branch

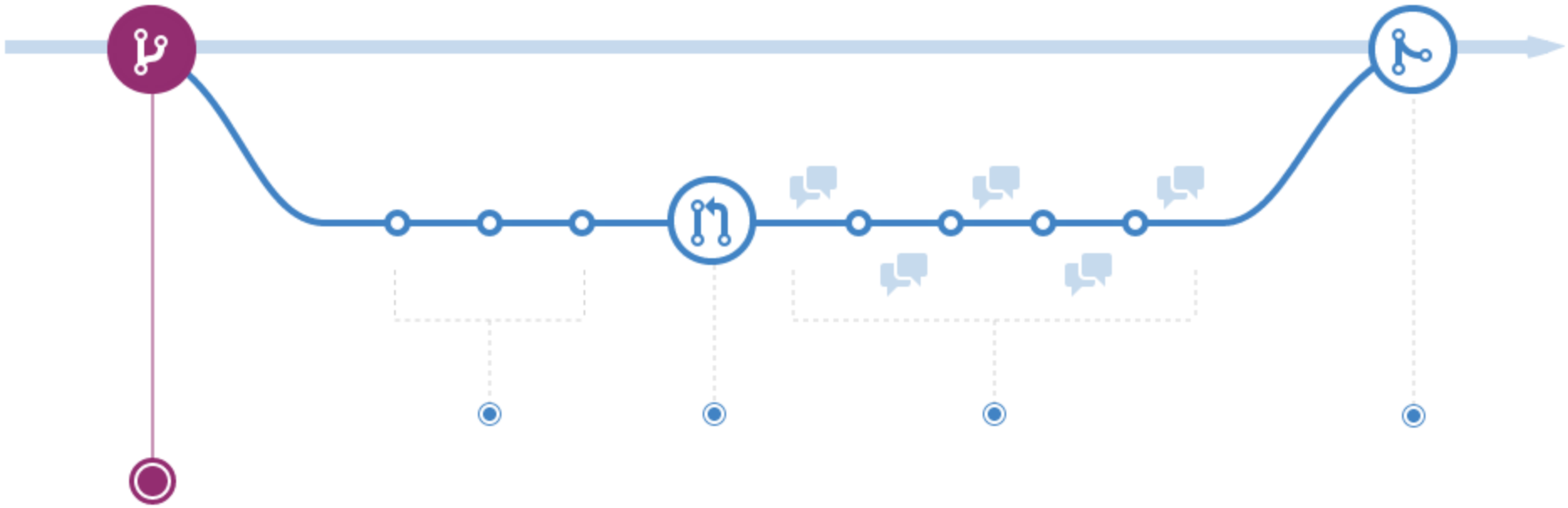
- It's good practice, to review code that is relied on to always be in good shape before it gets to the `main` branch.
- A **branching strategy** guides your team in working with version control effectively and consistently and lays out when code can be merged into the `main` branch.

An example of a branching strategy

- Use feature branches for all new features and bug fixes.
- Merge feature branches into the main branch using **pull requests**.
  - A **pull request** (sometimes called merge request) is a request for review and merging of your code into the `main` branch.
- Keep a high quality, up-to-date main branch.
- There are many more workflows and strategies. Most importantly, **be consistent and agree upon a strategy with your team**.
- Demo

Github Flow

- Interactive Github Flow



Some Git Best Practices

- Small, clean, logical commits; one thing at a time.
- Meaningful commit messages.
- Commit early, commit often; it does not have to be perfect.
- Double check what you are pushing to the remote repository: use a `.gitignore` file!

Other things to keep in mind

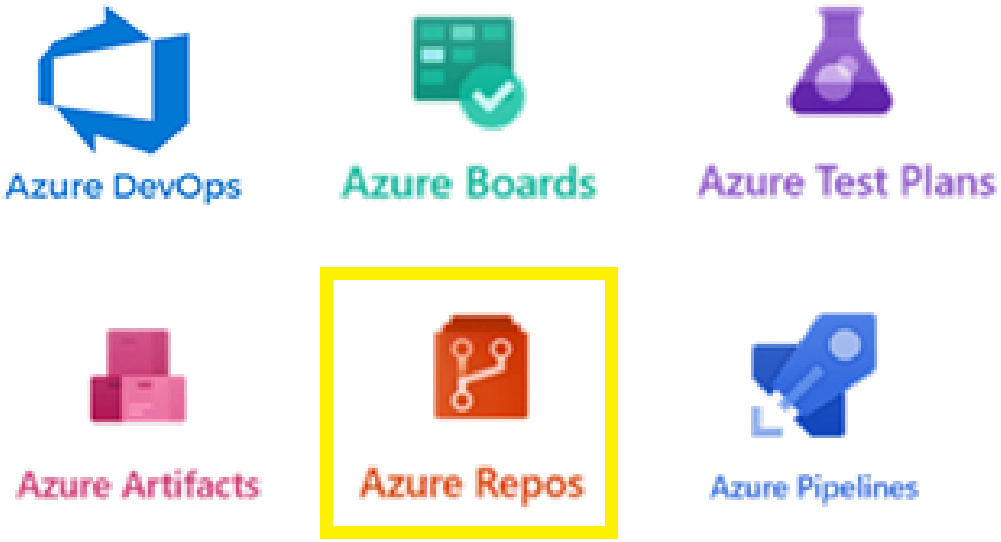
- You do not need to know everything by heart. Use Google and Stackoverflow 🙂!
- Cheat sheets can help.
- There is much more to Git than we covered today.
- Read up on the **Data Science Manifest** to learn about Best Practices when writing code.
    - Among version control, also concisely provides guidelines on coding practices, working together, testing and managing your project.

Git in DNB context

- Migrating towards Microsoft Azure Cloud.
- DNB is all into the Microsoft ecosystem:
  - Office (365): Word, Excel, Powerpoint, Outlook etc.
  - Skype, Teams, OneDrive, Sharepoint
  - Edge Browser
  - PowerBI
  - Azure

Git in DNB context

- Microsoft's offering of Git-based version control systems:
  - **Azure Repos**



Git in DNB context

- Only accesible within DNB: **Azure DevOps**.
  - Is accesible from the new Data Science Workspaces (DSW) which are replacing RAN(S).
  - There are too many Git alternatives at the moment: all Gitlab servers that exist now, e.g. in RAN(S), EOS, etc. will be replaced by **Azure DevOps Repos**.
- **Demo**

Your Turn

- Let's work on two cases where you describe the sequence of steps to solve the case.
- We'll break up into a number of breakout rooms, where you come up with a solution together. We'll get back together after each case to discuss the results.
- You can use the following cheat sheet with a list of commands that you might need.

Case I. Extending work of a colleague

> A colleague has written code that processes data you need for an analysis that you want to reproduce. He has checked in his code into version control on the `main` branch, in this case in an Azure DevOps Repository. You found some rows in your data that are clearly erroneous and should be filtered out. This means you want to change a part of your colleague's code.

Describe/write down the steps that you should take to obtain your colleague's code, make modifications that fix the bug and publish your changes to the `main` branch.

Case II. Extending work consistently

> Your colleague prefers to review what you have changed before you can push the changes to the `main` branch. He suggest a different workflow.

What would be an alternative workflow, i.e. a workflow that does not directly change the `main` branch?

What advantages does this workflow have?

List of commands

Commands	
`git fetch`	`git pull`
`git checkout -b <branch_name>`	`git commit -m <message>`
`git reflog`	`git download`
`git clone <url>`	`git stash`
`git init`	`git branch`
`git add *`	`git status`
`git push`	`git merge <branch_name>`

Any final questions?

Finally, please fill in the survey such that we can improve future workshops.

References and Resources

- Simple Git Guide
- Atlassian Git Cheatsheet
- Git Architecture
- Git Rebase
- Rebase vs Merge
- Azure DevOps Branching Strategies
- Git Reset