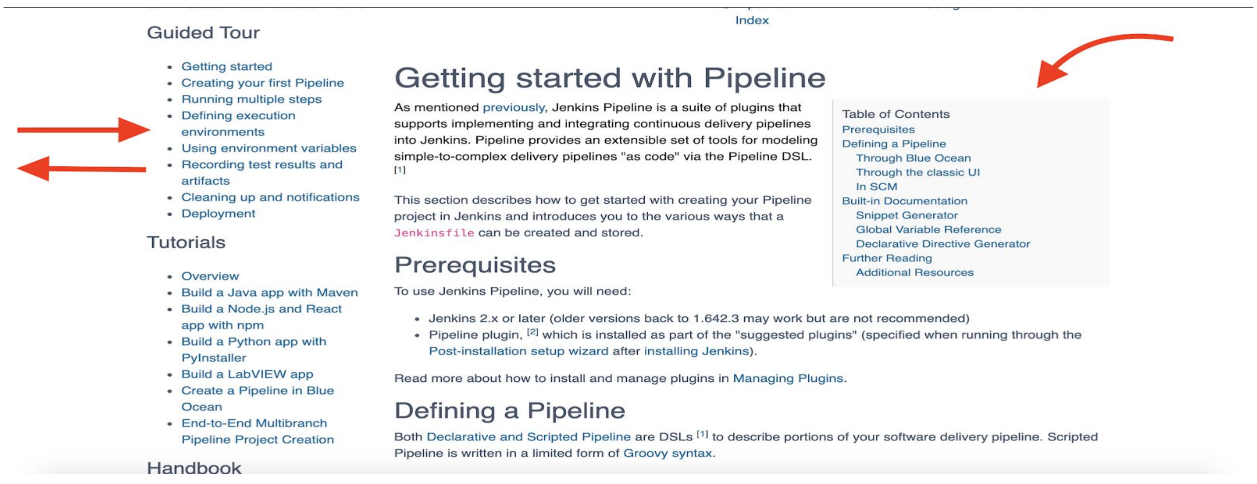
Jenkins User Handbook Feedback

Congestion:

The top of the pages seem to be predominantly bounded by the table of contents and the links leading to other tutorials and documentation, like "*Guided Tour*", "*Tutorials*" and "*Tutorial Blog Posts*", to name a few. This overuse of space makes the pages seem cluttered:



Index

Guided Tour

- Getting started
- Creating your first Pipeline
- Running multiple steps
- Defining execution environments
- Using environment variables
- Recording test results and artifacts
- Cleaning up and notifications
- Deployment

Tutorials

- Overview
- Build a Java app with Maven
- Build a Node.js and React app with npm
- Build a Python app with PyInstaller
- Build a LabVIEW app
- Create a Pipeline in Blue Ocean
- End-to-End Multibranch Pipeline Project Creation

Handbook

Getting started with Pipeline

As mentioned previously, Jenkins Pipeline is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins. Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline DSL. [1]

This section describes how to get started with creating your Pipeline project in Jenkins and introduces you to the various ways that a Jenkinsfile can be created and stored.

Table of Contents
Prerequisites
Defining a Pipeline
Through Blue Ocean
Through the classic UI
In SCM
Built-in Documentation
Snippet Generator
Global Variable Reference
Declarative Directive Generator
Further Reading
Additional Resources

Prerequisites

To use Jenkins Pipeline, you will need:

- Jenkins 2.x or later (older versions back to 1.642.3 may work but are not recommended)
- Pipeline plugin, [2] which is installed as part of the "suggested plugins" (specified when running through the Post-installation setup wizard after installing Jenkins).

Read more about how to install and manage plugins in Managing Plugins.

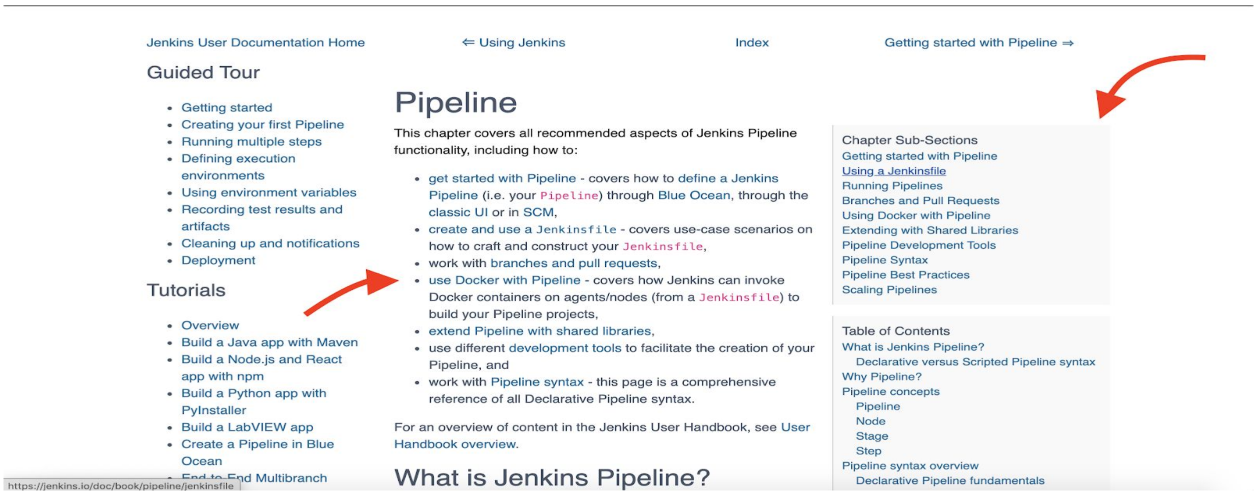
Defining a Pipeline

Both Declarative and Scripted Pipeline are DSLs [1] to describe portions of your software delivery pipeline. Scripted Pipeline is written in a limited form of Groovy syntax.

Extra space on the left and Table of Contents is placed in the middle of the content

As you can see from the arrows in the example above, the space is misused, thereby cluttering the essential information that the user needs to read. **Those links can be pushed further to the left, creating more space for the content.**

The Table of Contents can be placed at the top of the page, if required with the content following underneath it or could be placed further to the right, with a smaller font and size perhaps. The font size of these menu options and Table of Contents shouldn't be the same font size of the content:



Jenkins User Documentation Home ⇐ Using Jenkins Index Getting started with Pipeline ⇒

Guided Tour

- Getting started
- Creating your first Pipeline
- Running multiple steps
- Defining execution environments
- Using environment variables
- Recording test results and artifacts
- Cleaning up and notifications
- Deployment

Tutorials

- Overview
- Build a Java app with Maven
- Build a Node.js and React app with npm
- Build a Python app with PyInstaller
- Build a LabVIEW app
- Create a Pipeline in Blue Ocean
- End-to-End Multibranch

Pipeline

This chapter covers all recommended aspects of Jenkins Pipeline functionality, including how to:

- get started with Pipeline - covers how to define a Jenkins Pipeline (i.e. your Pipeline) through Blue Ocean, through the classic UI or in SCM,
- create and use a Jenkinsfile - covers use-case scenarios on how to craft and construct your Jenkinsfile,
- work with branches and pull requests,
- use Docker with Pipeline - covers how Jenkins can invoke Docker containers on agents/nodes (from a Jenkinsfile) to build your Pipeline projects,
- extend Pipeline with shared libraries,
- use different development tools to facilitate the creation of your Pipeline, and
- work with Pipeline syntax - this page is a comprehensive reference of all Declarative Pipeline syntax.

For an overview of content in the Jenkins User Handbook, see User Handbook overview.

What is Jenkins Pipeline?

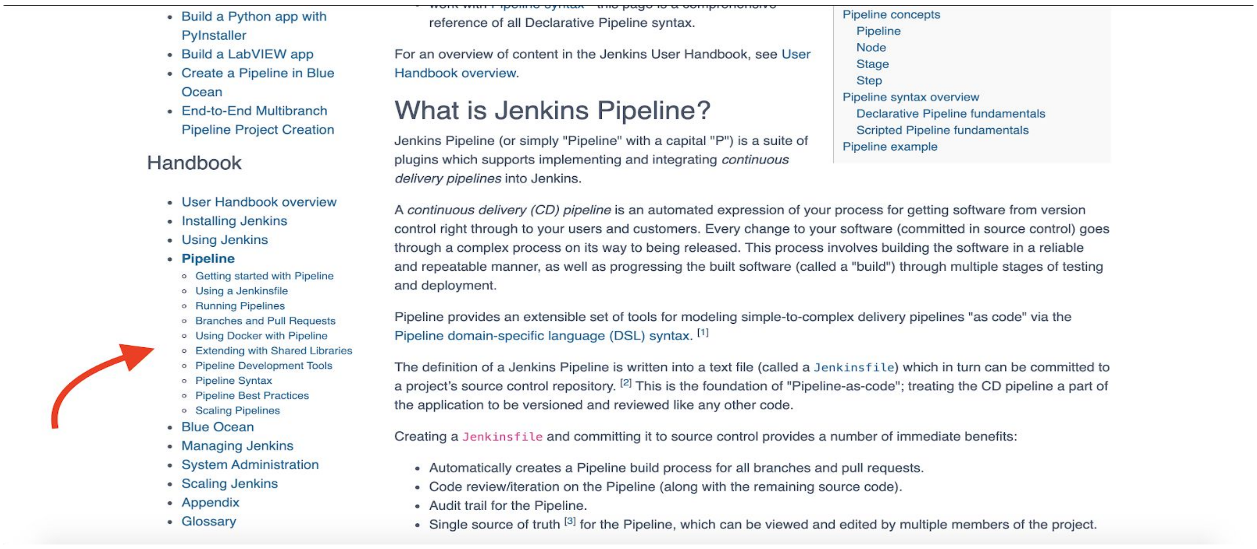
Chapter Sub-Sections
Getting started with Pipeline
Using a Jenkinsfile
Running Pipelines
Branches and Pull Requests
Using Docker with Pipeline
Extending with Shared Libraries
Pipeline Development Tools
Pipeline Syntax
Pipeline Best Practices
Scaling Pipelines

Table of Contents
What is Jenkins Pipeline?
Declarative versus Scripted Pipeline syntax
Why Pipeline?
Pipeline concepts
Pipeline
Node
Stage
Step
Pipeline syntax overview
Declarative Pipeline fundamentals

https://jenkins.io/doc/book/pipeline/jenkinsfile

Same links to the same content and font size is fully consistent throughout the web page

I feel that there **isn't a requirement for the Chapter Sub-Sections as all the links mentioned there already have the same links within the content.** The same links are in the menu options to the left as well:



- Build a Python app with PyInstaller
- Build a LabVIEW app
- Create a Pipeline in Blue Ocean
- End-to-End Multibranch Pipeline Project Creation

Handbook

- User Handbook overview
- Installing Jenkins
- Using Jenkins
- **Pipeline**
  - Getting started with Pipeline
  - Using a Jenkinsfile
  - Running Pipelines
  - Branches and Pull Requests
  - Using Docker with Pipeline
  - Extending with Shared Libraries
  - Pipeline Development Tools
  - Pipeline Syntax
  - Pipeline Best Practices
  - Scaling Pipelines
- Blue Ocean
- Managing Jenkins
- System Administration
- Scaling Jenkins
- Appendix
- Glossary

reference of all Declarative Pipeline syntax.

For an overview of content in the Jenkins User Handbook, see User Handbook overview.

What is Jenkins Pipeline?

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating *continuous delivery pipelines* into Jenkins.

A *continuous delivery (CD) pipeline* is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax. [1]

The definition of a Jenkins Pipeline is written into a text file (called a `Jenkinsfile`) which in turn can be committed to a project's source control repository. [2] This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.

Creating a `Jenkinsfile` and committing it to source control provides a number of immediate benefits:

- Automatically creates a Pipeline build process for all branches and pull requests.
- Code review/iteration on the Pipeline (along with the remaining source code).
- Audit trail for the Pipeline.
- Single source of truth [3] for the Pipeline, which can be viewed and edited by multiple members of the project.

Pipeline concepts
- Pipeline
- Node
- Stage
- Step

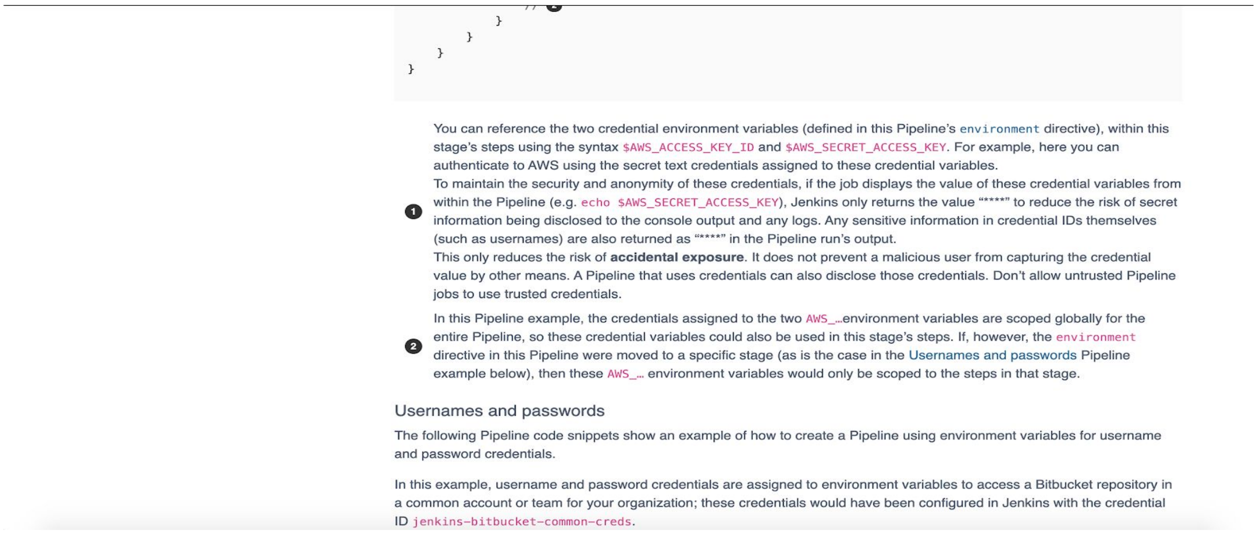
Pipeline syntax overview
- Declarative Pipeline fundamentals
- Scripted Pipeline fundamentals

Pipeline example

Another set of links to the same content within the same web page of documentation

Technical Jargon:

The guide seems like it was written specifically for someone who has been in the Software Engineering industry or has a strong working knowledge of Jenkins or someone who has used the documentation for quite awhile. There are a lot of parts in the documentation where topics are well explained, but even the explanation might take a beginner some time to understand.



```
                }
            }
        }
    }
```

1 You can reference the two credential environment variables (defined in this Pipeline's `environment` directive), within this stage's steps using the syntax `$AWS_ACCESS_KEY_ID` and `$AWS_SECRET_ACCESS_KEY`. For example, here you can authenticate to AWS using the secret text credentials assigned to these credential variables.
To maintain the security and anonymity of these credentials, if the job displays the value of these credentials from within the Pipeline (e.g. `echo $AWS_SECRET_ACCESS_KEY`), Jenkins only returns the value "****" to reduce the risk of secret information being disclosed to the console output and any logs. Any sensitive information in credential IDs themselves (such as usernames) are also returned as "****" in the Pipeline run's output.
This only reduces the risk of **accidental exposure**. It does not prevent a malicious user from capturing the credential value by other means. A Pipeline that uses credentials can also disclose those credentials. Don't allow untrusted Pipeline jobs to use trusted credentials.

2 In this Pipeline example, the credentials assigned to the two `AWS_...`environment variables are scoped globally for the entire Pipeline, so these credential variables could also be used in this stage's steps. If, however, the `environment` directive in this Pipeline were moved to a specific stage (as is the case in the Usernames and passwords Pipeline example below), then these `AWS_...` environment variables would only be scoped to the steps in that stage.

Usernames and passwords

The following Pipeline code snippets show an example of how to create a Pipeline using environment variables for username and password credentials.

In this example, username and password credentials are assigned to environment variables to access a Bitbucket repository in a common account or team for your organization; these credentials would have been configured in Jenkins with the credential ID `jenkins-bitbucket-common-creds`.

Clustered abundance of highly technical information

There is a lot of information that is grouped into one point at a time. It is slightly difficult to understand where one ends and the next begins. A lot of technical terms like "*accidental exposure*", "*global scope*" and "*bitbucket repository*" can be slightly confusing to a user who is trying to learn how to get started.

Perhaps adding a page for a glossary at the end would be helpful, if a lot of the technical terms have to be present in the documentation.

It also seems as though there is an abundance of information about one topic in a single page.

Maybe splitting the topics into more separate pages would help the user feel like they accomplished a steady amount of learning after each page, rather than feeling like the information they need to learn, is never ending.

Key Points:

- Too much technical information that may not be easily understood by a beginner to engineering or Jenkins in particular.
- Too much of unused space in the webpage, making each page of documentation seem cluttered.
- Unnecessary abundance of links across each page that link to the same content.