Table of Contents

[GETTING STARTED: 1](#_Toc53407546)

[NECESSARY AND RECOMMENDED READING: 1](#_Toc53407547)

[GETTING ACCESS: 1](#_Toc53407548)

[SOME BASIC INFO ON BITBUCKET: 1](#_Toc53407549)

[SOME BASIC INFO ON THIS DOCUMENT: 1](#_Toc53407550)

[USING GIT FROM A COMMAND PROMPT: 2](#_Toc53407551)

[CONFIGURING GIT FOR THE FIRST TIME: 2](#_Toc53407552)

[PUTTING CODE INTO A REPOSITORY: 2](#_Toc53407553)

[CLONING A GIT REPOSITORY ON BITBUCKET: 4](#_Toc53407554)

[GIT BASICS 5](#_Toc53407555)

[IMPORTANT COMMANDS 6](#_Toc53407556)

[SECTION A. BASIC COLLABORATION - WHEN YOU ARE SURE OTHERS ARE NOT WORKING ON THE REPOSITORY AT THE SAME TIME. 7](#_Toc53407557)

[CHECKLIST A 7](#_Toc53407558)

[CHECKLIST A WITH EXPLANATIONS 8](#_Toc53407559)

[SECTION B. SYNCHRONOUS AND FULL COLLABORATION - WHEN OTHERS MAY BE WORKING ON THE SCRIPT AT THE SAME TIME, AND WHEN YOU MAY WANT FEEDBACK ON A FEATURE: 9](#_Toc53407560)

[CHECKLIST B. 9](#_Toc53407561)

[WHAT IS REBASING? 10](#_Toc53407562)

[CREATING PULL REQUESTS: 11](#_Toc53407563)

[IMPORTANT OPTIONS 14](#_Toc53407564)

[TROUBLESHOOTING AND OTHER ACTIONS (under development) 14](#_Toc53407565)

# GETTING STARTED:

## NECESSARY AND RECOMMENDED READING:

Very important to read these articles, as they will save you substantial time:

1. *(Necessary reading)* Introduction to Git: <https://medium.com/@itswisdomagain/git-101-introduction-to-git-for-newbies-bb14f6f9fc1>
2. *(Necessary reading)* Introduction to GitHub (very similar to Bitbucket which is Skatteetaten's internal version control web-based hosting service): <https://medium.com/@itswisdomagain/github-101-introduction-to-github-for-newbies-efaf46c88406>
3. *(Recommended reading)* To understand the idea of **Basic Collaboration – Section A** in this document, you can read: <https://thenewstack.io/dont-mess-with-the-master-working-with-branches-in-git-and-github/>
4. *(Recommended reading)* To understand the idea of **Synchronous Collaboration – Section B** in this document, read: <https://www.atlassian.com/git/articles/simple-git-workflow-is-simple>

## GETTING ACCESS:

* You must apply for access to **UTV: Utvikler [L1] via ISIM Selvbetjening (Produksjon)** to get access to Bitbucket.
* Once approved to access Skatteetaten’s Bitbucket (you will receive an e-mail), you can start using Bitbucket.

## SOME BASIC INFO ON BITBUCKET:

Let’s cover a few basics about Bitbucket:

* To the left is a snapshot of an expanded side panel of Bitbucket, which is full of great options. You can expand or collapse the side panel at any time by clicking the << arrow at the bottom of it. Best to start learning with an expanded side panel however.
* Explore these, but do not worry if you do not know what each does. The webpage below and this workflow guide will help a little:

<https://bitbucket.org/product/guides/basics/bitbucket-interface>

* **Some jargon:** Repo is short for repository

## SOME BASIC INFO ON THIS DOCUMENT:

* This workflow is meant to standardize collaboration across Innsikt and Analysts in Skatteetaten. **It is not foolproof** and **is a work-in-progress**, so better workflows may exist and your feedback is appreciated. It was written as a collaborative initiative intended to help and maintaining it is not the author(s)’ responsibility. When you run into problems, first consult the trouble-shoot section, next google and if you find a better solution please do inform us.
* Color-guide to this document: This guide uses a command style for commands you shall type (and execute by pressing enter). It uses output for the terminal (e.g. Git Bash)’s output. It uses yellow highlighting and often <> around characters you must substitute for yourself.

## USING GIT FROM A COMMAND PROMPT:

Git Bash is a command prompt tool in Windows for using Git. There is a Git GUI or Graphical User Interface, however most Git users do use command prompt terminals, such as Git Bash. Follow the procedure below:

1. Install Git Bash from your Software Centre in your VDI Sikker (recommended) or Intern
2. Open it once installed
3. The interface is just as in any terminal, DOS, anaconda prompt, etc. Meaning that your usual DOS commands of pwd, dir, ls, mkdir, cd, cd .., all work. To learn the basics of navigating in a terminal, read <https://www.digitalcitizen.life/command-prompt-how-use-basic-commands>
4. By typing cd $Home, you will navigate to your C:/Brukere/<your-6-character-user-id, e.g. m87450>; Note that at any point in time you can type pwd to see which directory you are in.
5. Type cd Documents to navigate there
6. Type mkdir Git, to **m**a**k**e a **dir**ectory called Git
7. (Recommended) Get rid of MINGW64 by typing:

export PS1=”${PS1/\$TITLEPREFIX:}”; export PS1=”${PS1/\$MSYSTEM }” >> ~/.bashrc

1. Configure Git (explanation below) and then start Gitting

## CONFIGURING GIT FOR THE FIRST TIME:

You would need to configure Git by running the commands below (writer your own first and last name):

git config --global user.name "Surname, Firstname"

git config --global user.email "Firstname.Surname@skatteetaten.no"

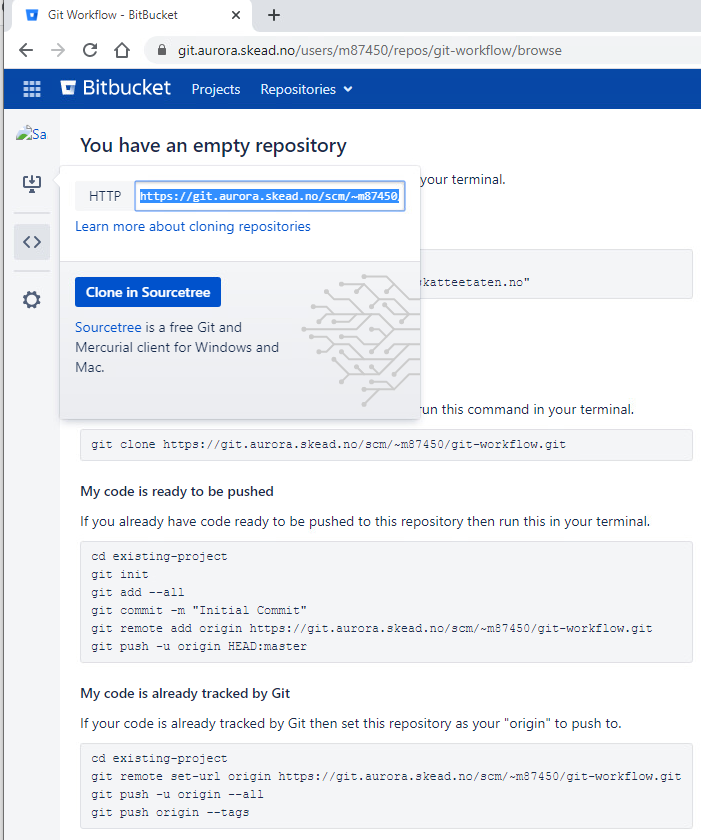
## PUTTING CODE INTO A REPOSITORY:

The person who will host the repository on their Bitbucket account, will follow these steps:

1. Create a folder where you place all the code, input files and output files into.

*Note: Best practice would be to place the input files in an input folder, and the output files in an output folder. The folder names could simply be INPUT or OUTPUT, but often names such as RAW DATA, ER (for EnhetsRegister), KLADD, etc make sense for input folders, and names such as VISUELLISERING, PRESENTAJSONER, can make sense for output files.*

1. Go onto your Bitbucket profile (<https://git.aurora.skead.no/profile>) and click on the "Create repository" rectangle. Then give it a Name and a Description. Best to leave default branch name empty so the default branch is called "master". Click on Create respository.
2. Find the repository's HTTP link. See the webpage popping up after the last step –snapshot below-, in which, there are four places from which to copy the repository's HTTP link. The left-hand pop-up window seen below will always exist for any repository, from which you can copy the highlighted HTTP link:



1. Once you are ready to push ("upload") your data to the repository, ensure you are in that folder by typing: cd existing-project and follow with the steps below (you can also find these commands above under "My code is ready to be pushed":
2. git init
3. Create a .gitignore file (which will help git ignore backup files created by editors, or intermediate files or any files you would like to keep private on your local Git folder), by following steps:
   1. touch .gitignore (this creates a .gitignore text file)
   2. nano .gitignore (this launches a text editor called Nano within Git Bash)
   3. Type appropriate file or folder names you would like Git to ignore, each on a line (for folders, you type ignoredfolder/ and for files you type file\_to\_ignore.extension; depending on what software your team codes in, here are great sample gitignore files: <https://git.aurora.skead.no/users/m87450/repos/gitignore/browse> e.g. for Python:

<https://git.aurora.skead.no/users/m87450/repos/gitignore/browse/Python.gitignore>)

* 1. Press CNTR+S to save and CNTR+X to quit (hint: ^means control in nano). ***Tip:*** *You can edit the text file after creating it in step a) by opening your file explorer and editing the file in notepad too.*
  2. **NOTE:** Once any file is added and tracked –i.e. included in git add, see next step-, it will be tracked and will not be ignored for anyone using the repository. So include files and folders you want ignored WHEN creating a depository OR for new files added or created, include them in the .gitignore file BEFORE running git add --all.
  3. **NOTE:** If you or someone HAD ADDED a file or folder, BEFORE it was included in the .gitignore text file, and you want to tell git to untrack it locally –i.e. untrack it for you only- then you can type git update-index --skip-worktree <path-name>. Each user will have to run this, to untrack it locally.
  4. **NOTE:** If you want to untrack the file for everyone using the repo, you can use option 1 of this stackexchange answer, but caution as it will delete files for others:

<https://stackoverflow.com/questions/936249/how-to-stop-tracking-and-ignore-changes-to-a-file-in-git/40272289#40272289>

* 1. **NOTE:** If you want to ignore select subfolders inside a shared (i.e. non-ignored) folder in a repository, you can specify the sharedfolder/subfolder/ inside the gitignore at the root of your directory, or you can place a .gitignore file within the sharedfolder where you add subfolder/ to it. This latter choice may be preferred in some cases.

1. git add --all
2. git commit -m "Initial Commit"
3. git remote add origin <COPY HTTP LINK HERE>
4. git push -u origin HEAD:master
5. Refresh the Bitbucket page, you should now see your folder uploaded in the remote repository called origin, on a branch called master.

## CLONING A GIT REPOSITORY ON BITBUCKET:

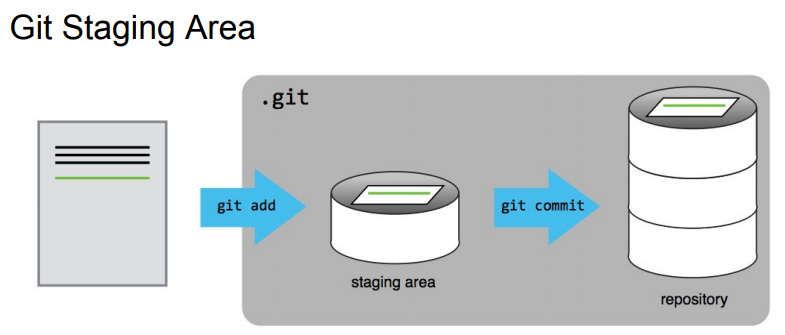
For the person who is not the main host of the repository, cloning the repository is the first action they take, so they can start collaborating on the project.

Although it's not a precise comparison, you can think of "Cloning a repository" as "Downloading the repository **for the first time**". The procedure for that is simply by copying the HTTP link that appears in photo above and typing:

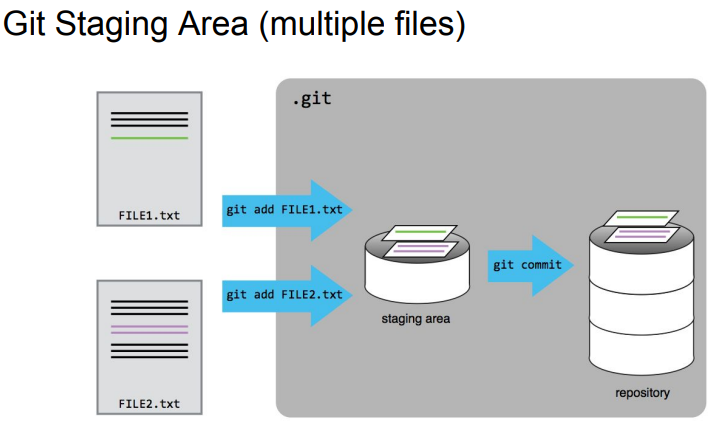
git clone <COPY HTTP LINK HERE>

## GIT BASICS

As a version control system, git takes your working directory (i.e. what you see in your file explorer or what you can see by typing ls in a terminal) and it will start tracking changes you make to it locally, but only if you add the changes! git add takes changes you save in your working directory into a “staging area”, and git commit –m “commit message label” takes your staged changes into your local repository and puts a label on it. Graphically speaking it looks like this:



The above graphic shows the staging and repository areas. The shaded .git rectangle in the graphic above, shows that a hidden folder in your working directory (created after you wrote git init), is now keeping an eye on the file you put into your staging area and what you then put into your repository. If you had multiple files, the graphic would look like this:



Why do we need a staging area before folders are in our local repository? Let’s take the answer from this [webpage](https://dev.to/sublimegeek/git-staging-area-explained-like-im-five-1anh):

*“Imagine a box. You can put stuff into the box. You can take stuff out of the box. This box is the staging area of Git. You can craft commits here. Committing is like sealing that box and sticking a label on it. The contents of that box are your changes. So, why not have the label mean something? You wouldn’t label a moving box with kitchen items as simply “stuff.”*

*As you make changes locally, Git can "see" them. However, figuratively speaking, they're out of the box. If you were to try and make a commit at this point, Git wouldn't have anything to commit.”*

## IMPORTANT COMMANDS

* At any point in time, in your Git Bash (or terminal) you can type git status, which will show you at what stage of the staging and committing process you are in, and give you guidelines on what to do next.
* You and your collaborators would and should add many different commits for changes you make. In order to see these from latest to earliest, just type git log, which will show you the commit names, hash codes (a unique number that git uses for each commit), date and author. When these grow long, your terminal will not show all until you press enter to scroll further down or press q to exit them.
* At any point in time, you can see which branch you are on, by looking at the (blue) colored branch name, which is by default always set to be (master). In Section A, you will see how you create new branches.
* You can always type git diff HEAD~1 to see the differences between the latest commit (called HEAD of the branch) and the change committed prior to that in the git branch you are in. Type git diff HEAD~2 to go see differences between the version going backwards two committed changes from the HEAD.
* You can use git diff to see differences between two branches too, e.g. git diff master..feature\_branch. **Hint:** For comparing to branches on the remote, when remote is called origin, then just type: git diff master..origin/master

# SECTION A. BASIC COLLABORATION - WHEN YOU ARE SURE OTHERS ARE NOT WORKING ON THE REPOSITORY AT THE SAME TIME.

**This is for collaborating when you are sure that others are not working simultaneously on the script! That is why it is called “Basic collaboration”, in that it is “not Synchronous Collaboration”.**

***“Don’t Mess with the Master” Checklist:***

This checklist (and this document) originated from summarizing this article:

<https://thenewstack.io/dont-mess-with-the-master-working-with-branches-in-git-and-github/>

***(pulling, branching, working, merging and pushing commands):***

## CHECKLIST A

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  | Be in the Git project directory you want to work on (it'll be created automatically if you "git clone") | Potential  Errors |
| 2 |  | git pull origin master |  |
| *3* |  | git checkout –b feature\_branch |  |
| 4 |  | *DEVELOP A NEW FEATURE, i.e. DO ALL THE CHANGES YOU WANT TO ADD A FEATURE WHICH HAS ONE OR MANY ADDED FUNCTIONALITIES. ASSIGN EACH ADDED FUNCTIONALITY AS ONE COMMIT ITERATIVELY (i.e. add an Atomic Commit). WHEN FINISHED, SAVE THE FILES & PROCEED AS BELOW* (Ensure you are in the same folder in the terminal):  *Hint:* In stages 5 or 6, you can type “git status” to see suggestions on what to do. |  |
| 5 (iterative) |  | git add --all |  |
| 6 (iterative) |  | git commit -m “git commit message” |  |
| 7 |  | git checkout master | .a |
| 8 |  | git fetch |  |
| 9 |  | git diff origin/master (to ensure no changes have been done to master while you were working) |  |
| 10 |  | git merge feature\_branch --no-ff and (optionally) write a merge commit message and ensure to exit (in Git Bash you exit by pressing esc, and entering :wq) |  |
| 11 |  | git push |  |
| 12 |  | git branch -d feature\_branch | .a, |

## CHECKLIST A WITH EXPLANATIONS

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  | Be in the Git project directory you want to work on | |
| 2 |  | git pull origin master | "Download" the latest version on the remote (called origin)’s master branch, and replace contents in my local Git folder |
| *3* |  | git checkout –b feature\_branch | Go to a branch called feature\_branch (if there is no such branch, this will create a new branch called feature\_branch) and start tracking changes in that branch. |
| 4 |  | DEVELOP A NEW FEATURE (See checklist above for more details) | |
| 5 |  | git add --all | You effectively add all the changes made into the staging area, ready to be committed (all this is happening on your local branch). |
| 6 |  | git commit -m “git commit message” | You commit the changes from the staging area into your local repository on your local branch. |
| 7 |  | git checkout master | You switch back to the local master branch (it’s only from here that you can merge your branch to the master) |
| 8 |  | git fetch | This fetches the latest remote (or origin) master repository into your local master branch. |
| 9 |  | git diff origin/master | A command to check if anything has changed in the master since you were working on your branch. If the terminal shows nothing, you know that the origin/master branch has not changed from your local master branch. If you see differences, then you will know that the master has changed; In this case, you will want to checkout your feature\_branch branch, and rebase your changes onto the new origin/master branch and only after ensuring everything works, merge your changes as per steps below. For rebasing, see steps 4-6 in checklist B in the Collaboration section of this document! |
| 10 |  | git merge feature\_branch --no-ff | You now merge your feature\_branch branch into your local master branch. To exit the message window that pops up where you can change the message, but the default message is fine, so you type “:wq” to exit.  *Note: --no-ff preserves feature history and easy full-feature reverts* |
| 11 |  | git push | You "upload" code from the local master to the remote master repository (on Bitbucket); you can now go to Bitbucket and check to see if the changes are made (It is a good idea). |
| 12 |  | git branch -d feature\_branch | Delete the feature\_branch branch by running this code. |

# SECTION B. SYNCHRONOUS AND FULL COLLABORATION - WHEN OTHERS MAY BE WORKING ON THE SCRIPT AT THE SAME TIME, AND WHEN YOU MAY WANT FEEDBACK ON A FEATURE:

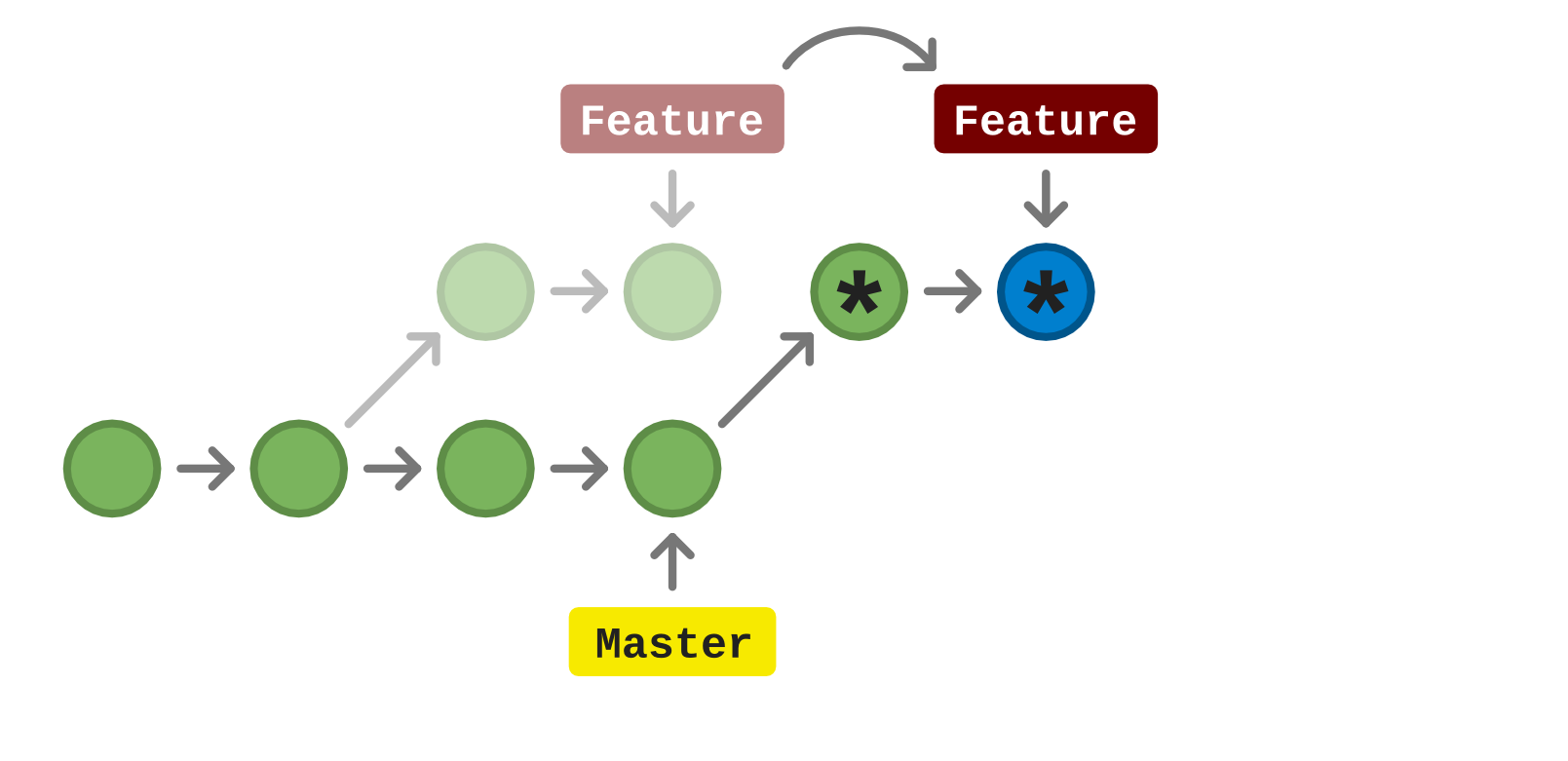
This follows Atlassian’s Simple Git workflow article: <https://www.atlassian.com/git/articles/simple-git-workflow-is-simple>

## CHECKLIST B.

|  |  |  |
| --- | --- | --- |
| 1 |  | Be in the Git project directory you want to work on (it'll be created automatically if you "git clone") |
| 2 |  | git pull origin master |
| *3* |  | git checkout –b feature\_branch |
| 4 |  | DEVELOP A NEW FEATURE: DO ALL THE CHANGES YOU WANT. ASSIGN EACH ADDED FUNCTIONALITY AS ONE COMMIT. SINCE *YOU KNOW OTHERS MAY BE WORKING ON THE MASTER, REGULARLY REBASE (rebasing is equivalent to step 7 below)!* |
| 5 |  | git add --all |
| 6 |  | git commit -m “git commit message” |
| 7  (iterative – pre-conflict-resolution) |  | git fetch origin  git rebase origin/master  RESOLVE ANY CONFLICTS COMING OUT OF THE REBASE!  YOU HAVE STARTED THE REBASING PROCESS. IF THERE ARE NO CONFLICTS, THE REBASING PROCESS IS FINISHED, GO TO STEP 8. OTHERWISE, RESOLVE ANY CONFLICTS COMING OUT OF THE REBASE ATTEMPT BY EDITING THE CONFLICTING FILES (Read the line saying CONFLICT (content): Merge conflict in <conflicted\_files>): THEN,  git add <conflicted\_files>  git rebase --continue |
| 7b  (conditional) |  | IF you want other people to help you develop the same feature branch, push it to remote by typing:  git push origin feature\_branch |
| 7c  (conditional) |  | IF you already had pushed the feature branch remotely, and you ARE working with other people on the same shared remote feature branch, then in addition to step 7, rebase changes coming from the remote feature branch by typing:  git rebase origin/feature\_branch  and then SOLVE ANY CONFLICTS. |
| 7d  (optional and conditional) |  | IF you want others to test your new feature, BEFORE it becomes merged with the remote master, then push this feature\_branch to Bitbucket, by typing:  git push –u origin feature\_branch  AND THEN CREATE A PULL-REQUEST. If you got feedback from your pull-request you can incorporate it and go back to the iterative step 7, and then 8. |
| 8 |  | MAKE SURE YOUR FEATURE IS DEPLOYABLE (i.e. IT WORKS) AND NO ONE HAS CHANGED THE REMOTE MASTER SINCE YOU REBASED (i.e. during the time you were resolving conflicts), BEFORE YOU MERGE AND PUSH IT TO THE REMOTE MASTER AS BELOW.  *Hint:* *You can refresh the repository commits page on Bitbucket to ensure no one has done this during the time you were rebasing.* |
| 9 |  | git checkout master |
| 11 |  | git merge feature\_branch --no-ff |
| 12 |  | git push |
| 13 |  | git branch -d feature\_branch |

## WHAT IS REBASING?

Rebasing is to change the base of the branch you are working on! It is used when the reference branch you are building on, e.g. master, has one or more commits after you had started branching off and developing your feature. The graphic below explains it best:

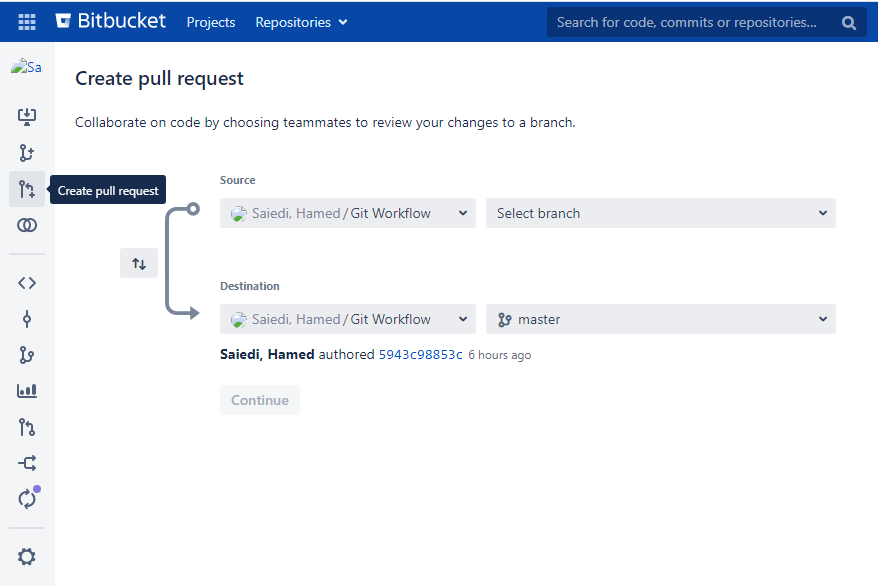


In the graphic above, each circle is a commit (read left to right). You can see that since the Feature branched off first, two more commits were implemented on the master branch. Well, we cannot merge and push our feature out directly now. So what we do, is we fetch it, then we rebase it (i.e. ask git to move our changes on top of the two new master commits). Git does this, but often conflicts can happen, which you would have to resolve. Fortunately, git tells you in which file the conflicts happen. Then you can go into the file (e.g. Python file, etc) and fix them, and then continue the rebase, until it is successful. The asterisks \* inside the circles above, show how the two commits of the feature branch moved on top of the latest master changes. Now your feature branch head is at the blue-asterisked circle. If you git checkout master and git merge it, then the Master head will move there too.

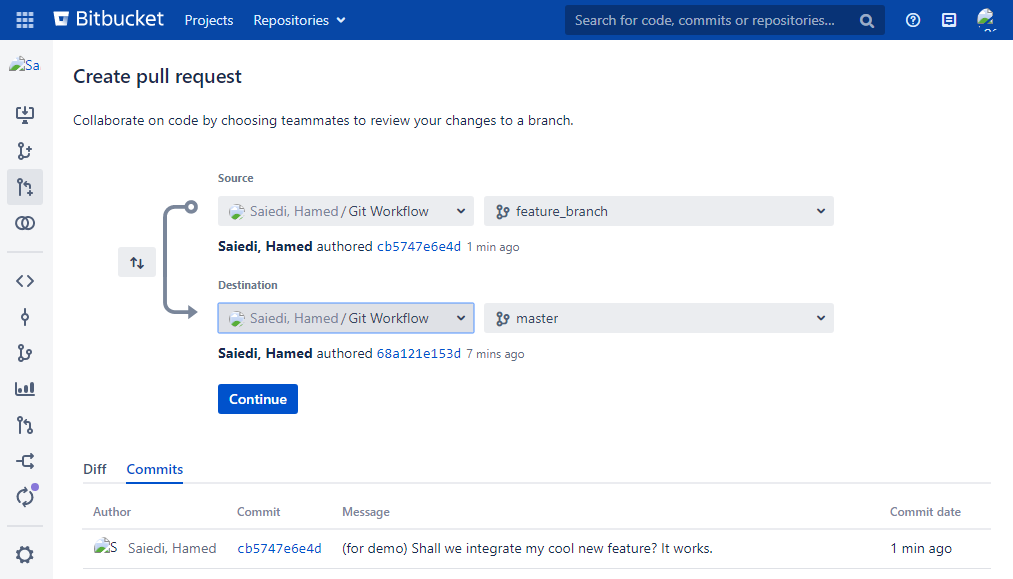
## CREATING PULL REQUESTS:

(The purpose of a pull request is to have the code maintainer(s) see your new feature and decide if they would like to merge it themselves, or give you the go-ahead to merge it if you have permissions)

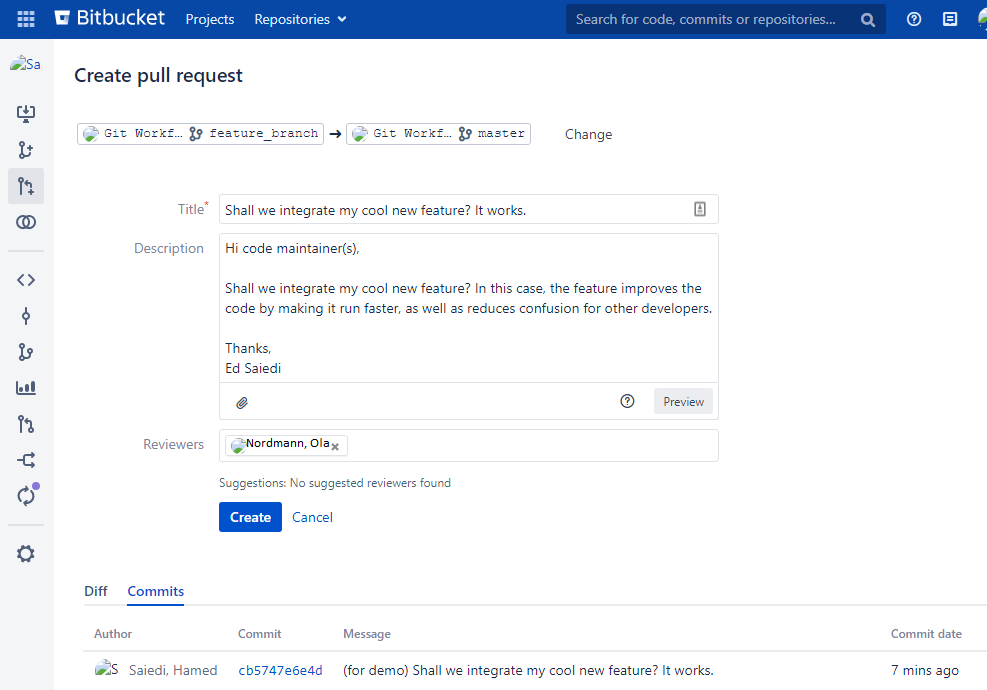
1. In the Bitbucket repository where you would like to make the pull-request, click on “Create Pull Request” (which is the icon that can be seen in the photo below).



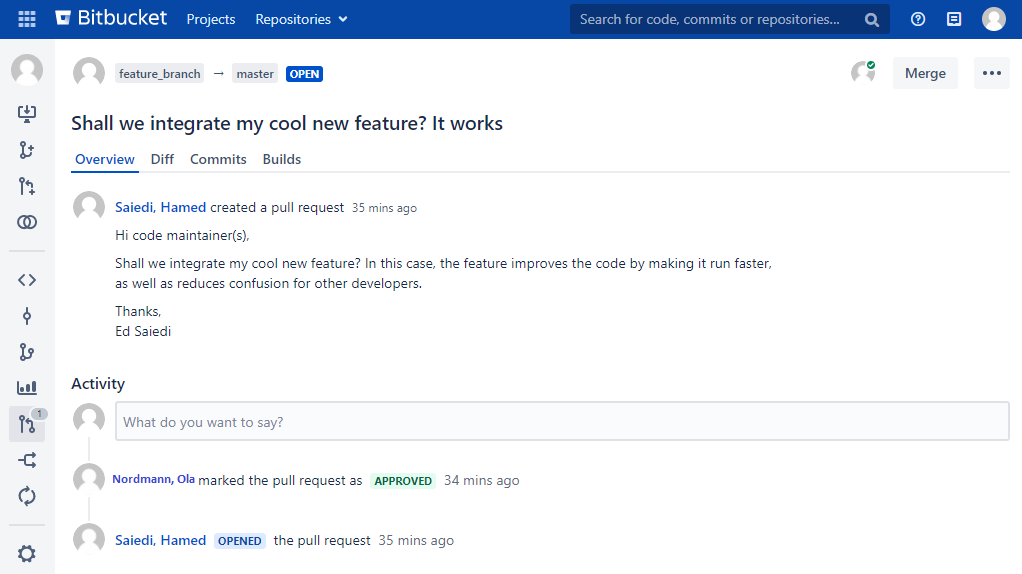
1. Clicking that will bring you to the snapshot shown above. Then simply select your branch (as in photo below; feature\_branch), click on Continue:



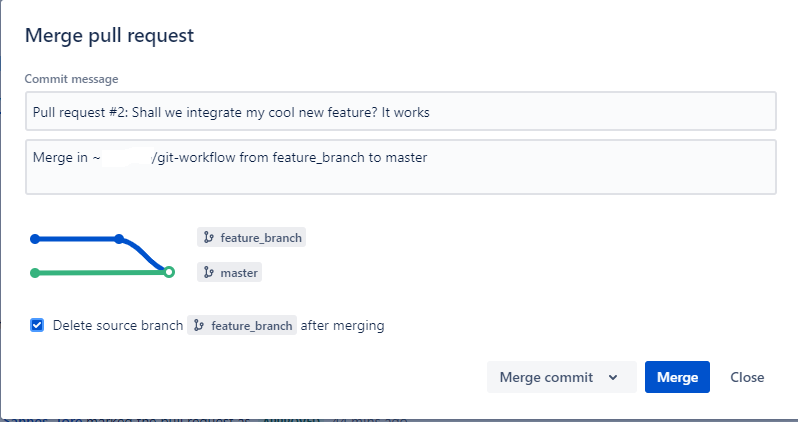
1. Now type a meaningful description and choose your desired reviewers (you must have had permissions) and click Create.



1. Now your chosen reviewer will receive an e-mail from [bitbucket-aurora@skatteetaten.no](mailto:bitbucket-aurora@skatteetaten.no) stating that you have created a pull request. The code maintainer/reviewer can then comment on your code, write feedback, highlight problematic lines, or choose to approve or merge your code. Be aware that if your code maintainer approves your code, it does not mean that (s)he has merged it (even though they have the choice of merging it).
2. Once the code maintainer has approved of your code, you will receive an e-mail, where you click “View Pull Request”. Now you can merge your feature, either by clicking merge (top-right of snapshot below; **make sure** to rebase if time has passed and others may have developed the master further, and ensure the rebased code is deployable) or by repeating steps 7-13 of the Section B Checklist.



1. You will be prompted with a dialogue box asking you whether to merge in the feature\_branch to the master. Let the “Delete source branch” option be checked (it ensures too many branches are left lying around and confusing developers, and is not that risky as the atomically-added commits in the feature\_branch will always be preserved)



1. Ensure you git pull origin master afterwards on your Git Bash. Congratulations. Now you are done.

Beware that if you do not do a pull request, when your feature needed a second test or set of eyes, your operations (Ops) team or code maintainers or collaborators may be have to deal with it:



# IMPORTANT OPTIONS

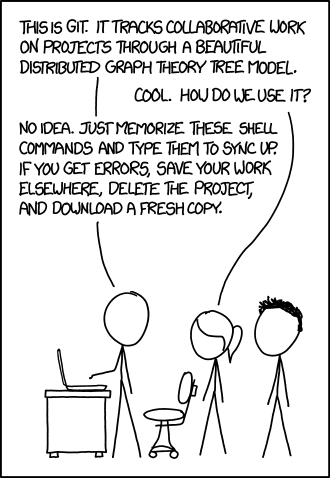
If you wanted to push your local branch directly to the remote, as a remote branch, you run this (after git add --all and git commit -m “<YOUR COMMIT MESSAGE>” and then git checkout master:

git push origin feature\_branch

(Hint: essentially all git push and pulls have this structure, whether the remote is called origin, developer, etc.: “git push <remote> <branch>”)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# TROUBLESHOOTING AND OTHER ACTIONS (under development)



**Troubleshooting 101: The above meme may look like a joke, but in fact when every other troubleshooting issue below fails, the solution ´stated above (saving your work elsewhere, deleting the project and downloading (i.e. git clone ing) a new project, works wonders!**

**Possible errors:**

If during any of the process in **WINDOWS**, you encounter this error: LF will be replaced by CRLF in git, use this:

git config --global core.autocrlf true

If you encounter this in **Linux or MacOS**, type this:

git config --global core.autocrlf input

(link with explanation is here: <https://stackoverflow.com/questions/5834014/lf-will-be-replaced-by-crlf-in-git-what-is-that-and-is-it-important>)

Hint: If it just gives a warning and not an error, then you may not have to run these, and may be able to continue your work without disruptions. The reason why this happens could be due to the git repo being pushed to by a Unix-user and a Windows-user, where end of sentences use different LF (Line Feeds) or CR (carriage returns). The stackoverflow link explains it best.

If you get the error:

git push origin master

fatal: Authentication failed for 'https://git.aurora.skead.no/scm/~m87450/git\_workflow.git/', it means that you had not configured properly OR your password was changed, then this is what to do:

git remote set-url origin https://<USERNAME>:<PASSWORD>@<COPY HTTP LINK WITHOUT https:// HERE, e.g. git.aurora.skead.no/scm/~m87450/git-workflow.git>

After changing your windows password, you might get this error when you try to clone a new repository:

git clone https://git.aurora.skead.no/scm/~m87450/git-workflow.git

Cloning into 'git-workflow'...

fatal: Authentication failed for 'https://git.aurora.skead.no/scm/~m87450/ git-workflow.git/'.

To reset your password go to Kontrollpanel -> Brukerkontoer -> Legitimasjonsbehandling.

Choose Windows-legitimasjoner, then git:https://git.aurora.skead.no.

Click on Rediger, type your new Windows password and save.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If you want to merge two remote branches, use the following:

git push origin feature\_branch:master --force

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In case there are multiple merge conflicts when you are on the local master branch and you do a “git pull origin master” (remember that git pull is a git fetch plus a git merge), then run this to update your local master branch to the origin master branch by force:

git reset --hard origin/master

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To simply merge a specific file from a branch (A) to another branch (B), ensure you are on the branch you want to merge to (i.e. B), then simply type:

Git checkout branch\_a UpdatedFileOnBranchA

Afterwards, just git add --all and git commit –m “message” as you wish.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If you try to do a git pull and get the following error:

Error:

Please commit your changes or stash them before you merge.

If you want to stash them, you can execute:

git stash -u

If you want to delete all your changes, you can type:

git reset --hard origin/master

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To force pull from merge and overwrite local files:

git fetch --all

git reset --hard origin/master

git pull origin master

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If you get an error that some of your files will be overridden, but you don’t mind that, run this:

git reset --hard origin/master

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If you run into an error of this type:

error: RPC failed; curl 56 OpenSSL

And you believe it is because the repository you are pulling was too large, then run this to increase the bandwidth:

git config http.postBuffer 524288000

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Checklist A. Step 7.a)

As a rule of thumb, you must only checkout another branch, after having committed any changes on your current branch. If you don’t you can get this error:

$ git checkout add\_simple\_collab\_workflow

Please commit your changes or stash them before you merge.

In these cases, you may wish to discard (i.e. get rid of permanently) or stash (i.e. put them aside for potential future retrieval) your changes.

If you want to discard them, you can execute:

git checkout -- .

If you want to stash them, you can execute:

git stash -u

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Checklist A. Step 12.a)

If your feature branch is ahead of your local master for any reason and the changes ahead are not integrated, you may get this error:

$ git branch -d feature\_branch

error: The branch 'feature\_branch' is not fully merged.

If you are sure you want to delete it, run 'git branch -D feature\_branch'.

Simply follow their instructions if you want to delete the branch, otherwise checkout if you had intended to merge your branch changes ahead of deleting it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*