**General Workflow:**

* If this is the very beginning start by cloning the repository
* Use “pull” to get the latest version
* If mob-programming, or parallel programming, and only one person is coding at a time, add changes to repo like this:
  + “git add .”
  + “git commit -m”@[USERNAME], [message about changes made or added].”
  + “git push”
* And when the next person starts programming, make sure to pull whatever the previous person has done before you start coding!
* If working on your own, you must work on a branch and not directly on master:
  + Create a new local branch: “git branch [new branch name]”
    - Local branches are still there even if you leave the project or git or whatever… it’s still there unless you delete it.
  + Switch to your branch: “git checkout [name of branch to switch to]”
  + Do work
  + If you want to add work you’ve done to your branch do a git add and then a git commit -m””
  + If you want to add your local branch to the repository do a git push
  + To keep your branch up to date with the master branch:
    - Make sure you are on your local branch
    - Type “git rebase origin master”
    - You are now up to date
  + If you feel ready with the work in your branch and wish to add it to the master branch:
    - Make sure your local master branch is up to date by doing:
      * make sure you are on the master branch
      * “git fetch origin master”
      * “git merge”
    - Make your local branch up to date with local master by doing a rebase
      * Switch to your local branch that you wish to rebase
      * “git rebase master”
    - Switch to the master branch
    - Type “git merge [name of the branch you wish to add to master]
    - Done!

**Clone Repo:**

### **Cloning an Existing Repository**

If you want to get a copy of an existing Git repository — for example, a project you’d like to contribute to — the command you need is git clone. If you’re familiar with other VCS systems such as Subversion, you’ll notice that the command is "clone" and not "checkout". This is an important distinction — instead of getting just a working copy, Git receives a full copy of nearly all data that the server has. Every version of every file for the history of the project is pulled down by default when you run git clone. In fact, if your server disk gets corrupted, you can often use nearly any of the clones on any client to set the server back to the state it was in when it was cloned (you may lose some server-side hooks and such, but all the versioned data would be there — see [Getting Git on a Server](https://git-scm.com/book/en/v2/ch00/_getting_git_on_a_server) for more details).

You clone a repository with git clone <url>. For example, if you want to clone the Git linkable library called libgit2, you can do so like this:

$ git clone https://github.com/libgit2/libgit2

That creates a directory named libgit2, initializes a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version. If you go into the new libgit2 directory that was just created, you’ll see the project files in there, ready to be worked on or used.

If you want to clone the repository into a directory named something other than libgit2, you can specify the new directory name as an additional argument:

$ git clone https://github.com/libgit2/libgit2 mylibgit

That command does the same thing as the previous one, but the target directory is called mylibgit.

Git has a number of different transfer protocols you can use. The previous example uses the https:// protocol, but you may also see git:// or user@server:path/to/repo.git, which uses the SSH transfer protocol. [Getting Git on a Server](https://git-scm.com/book/en/v2/ch00/_getting_git_on_a_server) will introduce all of the available options the server can set up to access your Git repository and the pros and cons of each.

**Making sure you’re up to date:**

### **Git pull**

It is recommended that you type ***“git pull”*** before you start working on your project. This will make sure that what you’re working on is the latest version of whatever branch or origin you’re working on in the repository. Doing this as the first thing you do minimizes the risk of encountering merge conflicts when you push what you’ve been doing to the repo.

**Preparing And Pushing Your Work:**

### **Git status**

To check if there are any changes to commit type ***“git status”***. This will display all files that are either untracked or that have changes in them.

### **Git add**

After this you can use two ways of adding untracked files or changes. Either use ***“git add .”*** to add all files and changes or use ***“git add [NAME-OF-FILE]”*** to add a single file. “Adding” them stages them to be committed.

### **Git commit**

Typing ***“git commit”*** prepares everything you’ve added to be pushed to the remote repository (eg. in most cases the repo on git-hub). Preferably you would type ***“git commit -m “[MESSAGE TO INCLUDE IN COMMIT]””***.

### **Git push**

Typing “git push” sends all you have committed to the repository.

**Managing branches:**

### **Why create a local branch?**

Working with branches is a good way to avoid merge conflicts when there a multiple people working on the same project. You will most likely encounter merge conflicts anyway, but if you’re working on your own local branch and then merge that branch with your local master branch, you can solve merge conflicts before pushing your code to the repo. If you would like to share your work in a branch without pushing it to the master there are also ways of creating a new branch in the repo to push your local branch to.

The naming distinction between local branches and branches in the repository is that a branch in your repository is called a “remote branch”.

There are other benefits of using branches as well. If you’d like to know more there are numerous resources online, not at the very gits own documentation.

### **Display branches.**

By typing ***“git branch”*** lists all local branches, ***“git branch -a”*** lists all branches including remote branches. The \*-sign denotes which branch you are currently on. What branch you are currently on can also be seen as a text colored in blue (ex: (myBranch)) in the text right above where you input your commands.

### **Adding a local branch.**

Type ***“git branch [BRANCH NAME]”***, to create a branch and switch to it at the same time type ***“git checkout -b [BRANCH NAME]”*** (do not do this if you only want to switch branch without creating a new one).

### **Switch local branch.**

Type ***“git checkout [BRANCH-NAME]”***

### **Delete local branch.**

Type ***“git branch -d [BRANCH NAME]”*** to do a “safe” delete. If there are currently adds or committs in the branch there will be complaints on deleting the branch. To force a delete type ***“git branch -D [BRANCH NAME]”***.

**!!Word of caution: forcing a delete of a branch will delete all currently committed or added data!!**

**Merge local branches**

Merging a branch means to put that branch’s history on top of another. It might be explained as “committing” your branch to another branch. It is not recommended to do this directly upon the remote repository. Focus on using this locally only unless you know what you’re doing. The beauty of git is that most things can be backed up and salvaged but some conflicts and errors can be more difficult to solve than others.

To merge branches place yourself on the branch onto which you want to merge another branch and type ***“git merge [BRANCH NAME]“.***

**Merging a local branch with the local master branch.**

A good workflow for doing this would be to first switch to your master branch, pulling your master branch to make sure you are up to date and then doing a merge from your local branch to your local master branch. This way you can spot and resolve merge conflicts before you push anything to the repository.

**Stashing:**

### **Why stash?**

Let’s say you’ve been working for a while but you want to do something else. Now you may not want to add your unfinished work to the repository but save it for later. This is where stash comes in. It lets you save your modifications of any files and save them for later, effectively removing them out of the equation when it comes to what state your branches are in. Or you could say that it lets you put aside any changes you’ve done since the last commit.

It can also be useful for when you’ve just started working and “OH SHIT! I forgot to do a pull!”.

### **How?**

Type “git stash” and voila!

**Getting what was stashed back.**

**Listing stashes.**

Typing “git stash list” shows all stashes.

**Removing stashes.**

Type “git stash drop [stash@{[STASH INDEX}]”

**Git expressions cheat-sheet.**

<https://gist.github.com/jedmao/5053440>

**Git workflow example.**

<https://gist.github.com/blackfalcon/8428401>

Stå på min branch

Rebasea till egen branch

Mergea in i master