**Branches - *branch early, and branch often***

Don't work in the master branch (except for simplistic like a readme), create a branch (copy of the repo) and work on that.

A branch is just a version of the local master repository. Whether there are differences between the branch and master depends on whether changes have been committed in the branch. Always commit changes before changing branches.

**Create and move between branches (checkout)**

Everything in Git is a reference/ pointer. Each time you checkout you’re making a pointer to working dir at a point in time.

Checkout moves between directories. It will modify the working directory to reflect the current state of the branch you moved to. The files that will be shown are dependent on what objects are indexed in the .git subdirectory.

**macoloco:test3 (master)$ git checkout [***<name>* **| master]** *To switch between branches*

**macoloco:test3 (master)$ git branch** *test**To create a branch (need to be in master)*

**macoloco:test3 (master)$ git branch** *To see a list of the branches*

*\* master*

*test*

**macoloco:test3 (master)$ git checkout** *test Switch to the new branch that was created*

*Switched to branch 'test'*

**macoloco:test3 (master)$ git checkout -b** *test1**Create and checkout a branch in one step*

*Switched to a new branch 'test1'*

You should always commit or discard your changes before switching branches so have a clean working directory.

**macoloco:test2 (test)$ git status** *Example of a clean working directory*

*On branch test*

*nothing to commit, working tree clean*

**Editing or adding files to a branch**

Updating files in a branch is a 2-part process, the changes first need to be indexed (***add***) and then referenced (***commit***).

Note that *git add* is used to add completely new files as well as to add modifications to files that already exist in the repo.

**macoloco:test2 (test)$** **git status** *Display the status of the working directory & index*

*On branch test*

*nothing to commit, working tree clean*

Updated or new files will show as *untracked.* This implies there are changes to the repo that haven’t been added to git.

**macoloco:test2 (test)$** **echo this is a test >** *test\_file**Makes a change to this file*

**macoloco:test2 (test)$** **git status**

*On branch test*

*Untracked files:*

*(use "git add <file>..." to include in what will be committed)*

*test\_file*

*nothing added to commit but* ***untracked files present*** *(use "git add" to track)*

**macoloco:test2 (test)$ git checkout** *test\_file**Reverts any changes made to that file*

Staging the file takes the changes from the working dir to the index. Is an orphan object as has not yet been referenced.

**macoloco:test2 (test)$** **git add** *test\_file or* ***git add .*** *adds all modified/ changed files*

**macoloco:test2 (test)$** **git status**

*On branch test*

*Changes to be committed:*

*(use "git reset HEAD <file>..." to unstage)*

*new file: test\_file*

**macoloco:test2 (test)$** **git reset HEAD** *test1* *Reverses the git add, so file is untracked again*

**macoloco:test2 (test)$** **git reset HEAD** *. Will reverse all the staged files*

For the object to be referenced (added to repository) and recorded permanently in version history it must be committed.

**macoloco:test2 (test)$** **git commit**  *Automatically opens vi to add a comment*

*[test e3b4680] Added TEST files - SH*

*2 files changed, 0 insertions(+), 0 deletions(-)*

*create mode 100644 test\_file1*

**macoloco:test2 (test)$ git commit -a** *No need for git add, it includes all modified files (however not new files)*

**macoloco:test2 (test)$ git commit -a -m** *“text” One-line commit message, skips opening the text editor*

**macoloco:test2 (test)$ git commit -s** *-s is an additional flag to say it was signed off by a specific user*

Combines a staged changed (*git add*) with the previous commit. Rather than editing the last commit it replaces it meaning it will be a new entity with its own ref. If nothing is staged it just amends the last commits comment

**macoloco:test2 (test)$ git commit --amend [-m** *<text>***]**

**Stash**

You should keep the working directory clean by not leaving untracked (uncommitted) files or changes in working directory when switching branches. Stash is a way to temporarily store uncommitted local changes keeping the working dir clean.

By default it only applies for untracked modifications, adding the -u flag makes it apply for untracked added files.

**macoloco:gitlab\_remote (master)$ git stash –help**

**macoloco:gitlab\_remote (master)$ echo hello > test1**

**macoloco:gitlab\_remote (master)$ git status**

*On branch master*

*Changes not staged for commit:*

*(use "git add <file>..." to update what will be committed)*

*(use "git checkout -- <file>..." to discard changes in working directory)*

*modified: test1*

*no changes added to commit (use "git add" and/or "git commit -a")*

**macoloco:gitlab\_remote (master)$ git stash** *Creates a stash with auto-generated message*

*Saved working directory and index state WIP on master: 5d3e11e check4*

**macoloco:gitlab\_remote (master)$ git status**

*On branch master*

*nothing to commit, working tree clean*

**macoloco:gitlab\_remote (master)$** **git stash -u** *Creates a stash for untracked files*

**macoloco:gitlab\_remote (master)$ git stash save** *"message"* *Creates a stash with customised* message

**macoloco:gitlab\_remote (master)$ git stash list** List all stashes. The latest is at the top with the index in {}

*stash@{0}: On master: change1*

*stash@{1}: WIP on master: 5d3e11e check4*

Can apply the stash in two ways, with either method if no index is quoted it uses the most recent stash from the list.

Pop removes stash from the stash list and applies it to the current working directory whereas apply does not remove it.

**macoloco:gitlab\_remote (master)$ git stash pop** *Removes and applies to most recent stash*

**macoloco:gitlab\_remote (master)$ git stash pop stash@{***1***}** *Removes and applies stash index 1*

**macoloco:gitlab\_remote (master)$ git stash apply** *Applies to most recent stash*

**macoloco:gitlab\_remote (master)$ git stash apply stash@{***2***}** *Applies stash index 2*

Can create a new branch with the latest stash or the specified stash index number

**macoloco:gitlab\_remote (master)$ git stash branch** *<name>* **stash@{***1***}**

**macoloco:gitlab\_remote (master)$ git stash drop stash@{***1***}** *Deletes the latest stash or quoted stash*

**macoloco:gitlab\_remote (master)$ git stash clear** *Deletes all the stashes in the repo*

**diff and show**

Diff shows changes before *git add* (staging), before *git commit* or between branches. This can be done for specific files or for all files within the branch. It only shows differences in the file contents, doesn’t show differences in directory structure.

**macoloco:test2 (test)$ git diff --help** *Are numerous options for this cmd*

**macoloco:test2 (test)$ git diff -- [***<file\_name>* **]** *Shows all the changes made before changes staged (git add)*

**macoloco:test2 (test)$ git diff HEAD -- [***<file >***]** *Shows changes from HEAD, so after git add but before git commit*

**macoloco:test2 (test)$ git diff master -- [***<file\_name>* **]** *Shows all the changes between master and branches*

**macoloco:test2 (test)$ git diff** *<commit\_hash>* *<file\_name>* *Differences from now (HEAD) to this commit*

**macoloco:test2 (test)$ git diff** *<commit\_hash>* *<commit\_hash>* *<file\_name>* *Differences between 2 commits*

**macoloco:test2 (test)$ git diff** *<src\_branch>* *<dst\_branch>* *Preview changes between branches*

**macoloco:test2 (test)$ git show** *Shows changes from the last commit*

**macoloco:test2 (test)$ git show** *00dfa72**Shows changes from a specific commit*

**Rollback untracked (staging) or committed changes**

Can too easily delete things with *reset*. Until comfy *git checkout* *commit* is a read-only way to go back and review what happened in the history of the project. Alternatively revert creates a new commit from current commit, nothing is deleted.

HEAD is the symbolic name for the currently checked out commit -- it's essentially what commit you're working on top of.

HEAD always points to the most recent commit which is reflected in the working tree.

**=rollback**

In case you did something wrong this cmd replaces local changes. It replaces the changes in your working tree with the last content in HEAD. Changes already added to the index, as well as new files, will be kept.

**macoloco:test2 (test)$ git checkout --** *<filename>*

If you instead want to drop all your local changes and commits, fetch the latest history from the remote server and point your local master branch at it:

**macoloco:test2 (master)$ git fetch origin**

**macoloco:test2 (master)$ git reset --hard origin/master**

**=checkout:**

Move between any commit with no loss of data in a repo. When you checkout a previous commit HEAD becomes detached, to get back to the last commit and reattach the HEAD checkout the original branch again.

**macoloco:project2 (test1)$** **git log --oneline**

*b9535b0 (HEAD -> test1) Change3 HEAD is always the current commit the repo is on*

*c0cef5d Change2*

*85f0d78 Change1*

*eb323fe (master) test This is the when branched from master*

**macoloco:project2 (test1)$** **git checkout** *85f0d78*

*Note: checking out '71743c9'.*

*You are in 'detached HEAD' state. You can look around, make experimental*

*changes and commit them, and you can discard any commits you make in this*

*state without impacting any branches by performing another checkout.*

*If you want to create a new branch to retain commits you create, you may*

*do so (now or later) by using -b with the checkout command again. Example:*

*git checkout -b <new-branch-name>*

*HEAD is now at 85f0d78 Change1*

**macoloco:test2 ((85f0d78...))$ git log --oneline**

*85f0d78 (HEAD) Change1*

*eb323fe (master) test*

**macoloco:test2 ((85f0d78...))$ git status**

*HEAD detached at 85f0d78*

*nothing to commit, working tree clean*

**macoloco:test2 ((85f0d78...))$ git checkout** *test1**Re-checkout the branch to reattach the HEAD*

*Previous HEAD position was 85f0d78 change1*

*Switched to branch 'test1'*

**macoloco:project2 (test1)$**

**macoloco:test2 (test)$** **git checkout -b** *<new\_branch>**71743c9 Restore version to new branch*

**macoloco:test2 (test)$** **git checkout** *71743c9 <file> Rollback just a certain file from this commit*

Can move around using **^** to go back one commit or **~** to specify the number of commits to move back.

**macoloco:test2 (test)$** **git checkout master^** *Move one commit back from HEAD of master*

**macoloco:test2 (test)$** **git checkout HEAD^** *Move one back from the HEAD of current branch*

**macoloco:test2 (test)$** **git checkout** *test***~2** *Move 2 commits back from HEAD of test*

Drops all modifications of foo.rb and replaces the file with its ~2 = two commits before the current.

**macoloco:test2 (test)$** **git checkout HEAD~2** *app/models/foo.rb*

**=revert:**

Changes all the files for the specified commit back to their state before that commit was completed. Nothing is deleted, a new commit is created with the included files reverted to their previous state.

**macoloco:test2 (test)$ git revert** *<commit\_ID>*

**macoloco:test2 (test)$ git revert** *<commit\_ID> <file\_name>* *Specify a file to revert*

**=reset:**

More dangerous than revert as git discards any commits between the current state of the repo and target commit.

**macoloco:test2 (test)$ git reset --hard**  *Undo all the changes you’ve introduced since the last commit*

**macoloco:test2 (test)$ git reset --hard** *<file\_name>* *Specify a file to reset*

**macoloco:test2 (test)$ git reset --hard** *<commit\_ID> Rewind your HEAD branch to the specified version*

**Merge**

Committed branch changes are only added to the master once the local branch is merged with the master. The merge cmd is run from the master branch quoting the branch to be merged. Once merged the old local branch can be deleted.

**fast-forward:** Simple situations where the master hasn’t changed but local branch has. Changes are added to the master.

**merge-commit:** More complex situations where there have been changes to both the local and master branch (source and destination branches). A full-blown merger takes the changes from both sides (local and master) and ties them together.

**macoloco:test2 (test)$ git checkout master**

**macoloco:test2 (master)$ git merge** *test*

*Updating edf5540..6b83be8*

*Fast-forward*

*test\_file | 1 +*

*test\_file1 | 0*

*test\_file2 | 0*

*3 files changed, 1 insertion(+)*

*create mode 100644 test\_file*

*create mode 100644 test\_file1*

*create mode 100644 test\_file2*

**macoloco:test2 (master)$ git branch -d** *test**Once a branch has been merged with master should delete it*

*Deleted branch test (was 6b83be8).*

If it hadn’t been merged git will warn you when you move branches and require **-D** if you try to delete it

**macoloco:test2 (test)$ git checkout master**

*Switched to branch 'master'*

*Your branch is ahead of 'origin/master' by 3 commits.*

*(use "git push" to publish your local commits)*

**macoloco:test2 (master)$ git branch -d** *test1*

*error: The branch 'test1' is not fully merged.*

*If you are sure you want to delete it, run 'git branch -D test1'.*

If changes have been made on the master since you last checked it out (for example other branches merged into it) you can merge the master into branch.

**macoloco:test2 (test)$ git merge master**

With *git pull* and *git merge* git it tries to auto-merge changes. Unfortunately, this is not always possible and results in conflicts. You are responsible to merge those conflicts manually by editing the files shown by git.

**macoloco:test2 (master)$ git add** *<filename>* *After changing, you need to mark them as merged*

**macoloco:test2 (master)$ git diff** *<source\_branch> <target\_branch>* *Before merging changes, you can preview*