**Git Notes**

Here are some common git operations for which we often find ourselves searching.

**Checkout a remote branch ‘develop’ and keep it linked to the origin**

git checkout -t origin/develop

It should report: “Branch develop set up to track remote branch develop from origin.”

**List all branches (both local and remote)**

git branch -a

**List local branches and their corresponding upstream remote**

**branches**

git branch -vv

**Revert local changes to a file (prior to commit)**

git checkout file

**Revert all local changes to the current subtree (prior to commit)**

git checkout .

**Cute hack to revert all local changes (prior to commit)**

git stash git stash drop

**Undo a commit**

git reset --soft HEAD^

**Undo multiple commits**

git reset --soft HEAD@**{**2**}**

Where 2 is the number of commits to undo.

**Update to latest HEAD, preserving local changes and local commits on top**

git stash

git pull --rebase

git stash pop

**Push changes on master to origin**

git push origin master

**Delete untracked files and directories**

git clean -df

**Interactively stage patches from changed file(s)**

git add -p &lt;path&gt;

**Roll back the last commit’s changes to a particular file**

git checkout HEAD@**{**1**}** – file

**Branches**

**Branch master to a new local branch “new\_branch”**

git checkout -b new\_branch master

**Push local branch to remote**

git push -u origin new\_branch

**Make the current local branch start tracking a corresponding remote branch**

git branch --set-upstream-to origin/new\_branch

This is not necessary if you used git push -u as suggested above.

**List the local branches that have already been merged to this one**

git branch –merged

**Diff a file between two branches**

git diff branch1 branch2 – file

**Delete a branch both locally and remotely**

git branch -rd origin/branch\_to\_kill

git branch -d branch\_to\_kill

git push origin :branch\_to\_kill

**Move a commit from bad\_branch to good\_branch**

*# First cherry-pick the commit onto the correct branch:*

git checkout good\_branch

git cherry-pick deadbeef

*# Then remove the commit from the bad branch:*

git checkout bad\_branch

git rebase -i

*# Change the undesirable commit to "noop"*

For more on branching, see [Git topic branches](https://imagej.net/develop/git/topic-branches).

**Rename the current branch**

git branch -m <new branch name>

**Git + SVN**

**Clone an SVN repository to a local Git repository**

git svn clone -s http://svn.code.sf.net/p/jhotdraw/svn/

**Commit and push changes, even with local changes in the working copy**

git commit git stash git svn dcommit git stash pop

**Update to latest trunk, preserving local changes and local commits on top**

git stash git svn rebase git stash pop

**Searching**

**Recursively search for HelloWorld.file (and display the most recent commit modifying it)**

git ls-tree -r HEAD | grep HelloWorld.file

**Recursively search for all files containing the phrase ‘import HelloWorld’**

git grep 'import HelloWorld'

**Recursively search for all files *in any topic branch* containing the phrase ‘import HelloWorld’**

git grep 'import HelloWorld' $(git rev-list --all --no-walk)

**History**

**Display a log with colored word diffs**

git log -p --color-words

Add -S to less to virtually wrap long lines.

**Display a diff with colored words between a file in one commit and a file in another commit**

git diff <commitA>:<file> <commitB>:<file> --color-words

Add -S to less to virtually wrap long lines.

**Display all contributing authors of a project including their e-mail**

git log --format**=**'%aN <%ae&>' | sort -u

Respects .mailmap.

**Viewing the history for a single file**

git log --follow HelloWorld.file

This history is algorithmically calculated and must be carefully preserved.

Simultaneous (within a single commit) significant changes + file renaming (including relocation) can prevent the algorithm from successfully tracing the file’s history, or cause it to begin tracing the wrong file.

Keeping code changes separate from renames should prevent this confusion, but it is good practice to check log --follow before pushing to a remote repository.

**See commits in branch B not present in branch A**

There are two main options. The first:

git log A..B

will display the different commits in full git log format. NB: the .. between commits is important to sure only the difference in commits is considered.

The second:

git cherry -v A B

will display a simple list of the different commits, one per line, with commit message and hash.

**Scripts**

There are some Git-related scripts available in the [scijava-scripts](https://github.com/scijava/scijava-scripts) project.

**List information about all remote branches including last author, commit date and unmerged commit count**

$SCIJAVA/bin/remote-branch-info.sh

**Advanced and/or dangerous**

**Create a repository with g+w permissions**

git init --shared**=**group

Or for a bare repository:

git init --bare --shared**=**group

(Bare repositories are meant for a remote server repository that all your coworkers push into and pull/fetch from.)

**Push all remote branches from one remote (e.g., “origin”) to another (e.g., “github”)**

git push github $(git **for**-each-ref refs/remotes/origin | \

grep -v HEAD | \

**while** read sha1 type ref

**do**

echo $ref:refs/heads/**${**ref#refs/remotes/origin/**}**

**done**)

**Another way to push all remote branches between remotes**

eval git push github $(git **for**-each-ref | \

sed -n 's/.\*\t\(refs\/remotes\/origin\/\(.\*\)\)$/\1:refs\/heads\/\2/p')

**Fully garbage collect and compact the repository (deletes all orphaned refs!)**

git reset --hard git **for**-each-ref --format**=**"%(refname)" refs/original/ | \

xargs -n 1 git update-ref -d git reflog expire --expire**=**now --all git gc --aggressive --prune**=**now

**Rewriting history**

**Split a subdirectory into a separate git repository**

See these posts on Stack Overflow:

* [Detach subdirectory into separate Git repository](http://stackoverflow.com/questions/359424/detach-subdirectory-into-separate-git-repository)
* [Detach subdirectory (that was renamed!) into a new repo](http://stackoverflow.com/questions/6638019/detach-subdirectory-that-was-renamed-into-a-new-repo)
* [Split large Git repository into many smaller ones](http://stackoverflow.com/questions/3910412/split-large-git-repository-into-many-smaller-ones)

**Throw away git-svn-id metadata**

git filter-branch --msg-filter ' sed -e "/^git-svn-id:/d" '

**Combine the first two commits of a Git repository**

See this post on Stack Overflow:

* [How do I combine the first two commits of a Git repository?](http://stackoverflow.com/questions/435646/how-do-i-combine-the-first-two-commits-of-a-git-repository)

**Change the author of a commit**

git commit --amend --author**=**"Author Name"

**Change the author of many commits**

See this post on Stack Overflow:

* [How do I change the author of a commit in git?](http://stackoverflow.com/questions/750172/how-do-i-change-the-author-of-a-commit-in-git)

**Merge multiple repositories**

See these posts on Stack Overflow:

* [Combining multiple git repositories](http://stackoverflow.com/questions/277029/combining-multiple-git-repositories)
* [git: Retroactively introduce several merges](http://stackoverflow.com/questions/4039682/git-retroactively-introduce-several-merges)

**Tutorials**

**Creating a shared remote repository**

ssh you@server

mkdir repos/remote.git

cd repos/remote.git

git --bare init --shared**=**group

logout

cd ~/local

git remote add origin ssh://you@server/home/you/remote.git

git push origin master

git config branch.master.remote origin

git config branch.master.merge refs/heads/master

Creates a bare remote repository at [ssh://server/home/you/remote.git](https://imagej.net/develop/git/Ssh___server_home_you_remote.git) that tracks your local repository in /home/you/local. Adopted from [Tim Lucas](http://toolmantim.com/articles/setting_up_a_new_remote_git_repository).

**Displaying a filtered set of commits**

Assume you want to see commits in branch stephan, but only those that are not part of the history of branch saalfeld:

git log stephan ^saalfeld

More realistically, if you want to see all the commits which are in a topic branch, but not yet merged into master:

git log --all ^master

If you want to see the changes which come from a topic branch which was merged in commit deadbeef, use this command line:

git log deadbeef^..deadbeef^2

Explanation: deadbeef is a merge commit, so its first parent (deadbeef^, can also be written as deadbeef^1) was the current HEAD when the merge was performed, and the second parent (deadbeef^2) is the tip of the branch which was merged. The argument A..B is short form of ^A B, i.e. all commits reachable from B excluding those which are also reachable from A.