Git cheat sheet

Stage changes

git add *filename*
Add file to staging area
git rm *filename*
Remove file from working area and staging
git mv *oldname newname*
Rename ("move") file in working area and index

Status

git status
Check working directory, index, etc
git log
View commit history, most recent at top.
<Space> = next page, b = back, q = quit
git log --graph
View commit history, with ASCII graphics

Commit

git commit -m "My message"
Commit staged files
git commit -a -m "..."
Stage all changes (in tracked files) and commit

Check out

git checkout *mybranch*
Switch to named branch
git checkout *mytag*
Switch to tagged commit. Will leave you in detached HEAD state
git checkout *09bbf3*
Swtich to specifc commit. Will leave you in detached HEAD state

Branch

git branch *mybranch*
Create new branch (but stay here)
git checkout *mybranch*
Switch to named branch
git checkout -b *mybranch*
Create new branch and switch to it
git branch
List (local) branches
git branch -a
List all branches (local and remote)
git branch -v
List all branches (verbosely)
git branch -d *mybranch*
Delete branch

Remotes

git remote
List remote repos
git remote -v
List remote repos with their URLs
git remote add *name url*
Add a remote. Colleagues' repos will be at file:///data/src-b0/SRCUSERID/universe-development
git remote remove *name*
Remove (your local reference to) remote
git remote rename *name newname*
Rename a remote
git remote set-url *name newurl*
Change the URL of a remote

Merge

git merge -m "..." *otherbranch*
Merge from other branch, and commit with message
git merge --ff-only otherbranch
Fast-forward current branch to catch up with other branch
git checkout *mybranch*
Switch to named branch
git checkout -b *mybranch*
Create new branch and switch to it
git branch
List (local) branches
git branch -a
List all branches (local and remote)
git branch -d *mybranch*
Delete branch

Merge conflicts

git status
Check what's conflicted
edit myfile
Edit each conflicted file to resolve the conflict
git add *myfile*
Signal your resolution by staging the file
git commit -m "Merged from..."
Commit your changes (with commit message) to complete the merge
git merge --abort
Abort problem merge and return to pre-merge state

Tag

git tag
List all tags
git tag *mytag*
Create a new tag here (at HEAD)
git tag -d *mytag*
Delete named tag
git push origin ----
Push a tag to remote repository

Push and pull

git push origin
Push branches (already known by origin) to origin
git push origin *mytag*
Push given tag to origin
git push --tags origin
Push all tags to origin
git fetch origin
Fetch all branches and tags from origin.
git merge *remname/brname*
Merge in from remote branch, e.g. origin/master
git pull origin
Fetch + merge. Only merges to current branch, and only if remote branch is linked to current branch (i.e. is a "remote tracking" branch)
git pull --rebase origin
Fetch + rebase instead of the default fetch + merge

Rebase

git rebase *somebranch*
Rebase current line of commits onto the end of named branch. You remain on the same branch, which now has a different history
git merge --ff-only *otherbranch*
Fast-forward current branch to catch up with other branch. Useful if you've rebased onto the branch you actually want to be on.

Rebase conflicts

git status
Check what's conflicted
edit myfile
Edit each conflicted file to resolve the conflict
git add *myfile*
Signal your resolution by staging the file
git rebase --continue
Continue the rebase after resolving the conflicts
git rebase --skip
Skip this commit in the rebase. Useful if you accept the original commit as-is and git complains you've not resolved anything
git rebase --abort
Abort problem rebase and return to initial state

Diff

git diff
Diff working directory against index
git diff HEAD
Diff working directory against HEAD (last commit). You can also name a branch, tag or commit with ID
git diff -- *filename*
Diff file against staged copy
git diff HEAD -- *filename*
Diff file against HEAD (last commit). You can also name a branch, tag or commit with ID

Push per feature

This strategy pushes up your changes per feature, always ensuring a straight-line history. You only ever develop on master.

git pull --rebase origin
Rebase your current work onto the end of latest changes from origin/master
git push origin
Push this all up to origin.

Push per release v1

This strategy has you unifying your team's changes in your local repository and then pushing them all up together. It always ensures a straight-line history. You develop on named release-branches.

Note that with this version, if you need to resolve conflicts it may look as if some of your resolutions have become lost. That's due to the order of the rebasing. It will all come together in the end

git checkout *rel-123*
Make sure you're on your release branch
git pull --rebase *bob rel-123*
Rebase your work onto the end of the latest changes from a team member. Repeat for all team members.
git pull --rebase origin master
Rebase your combined work onto the end of the latest changes from origin/master
git checkout master
Switch to master
git merge --ff-only *rel-123*
Bring it up to date with your team's changes
git push origin
Push this latest master up to the central server.

Now your other team members must forget their branches and pull down this latest version of master:

git checkout master
Switch to master
git pull --ff-only origin
Get the latest master from the origin

Recover files

These commands will all stage the file, too

git checkout *filename*
Recover file from staging
git checkout HEAD -- *file*
Recover file from last commit on this branch
git checkout f9b003 -- file
Recover file from specific commit

Push per release v2

This strategy has you unifying your team's changes in your local repository and then pushing them all up together. It always ensures a straight-line history. You develop on named release-branches.

This version avoids the "lost resolutions" effect of v1 because it rebases branches the other way round. On the other hand it has more steps and will periodically put you into detached HEAD state.

git checkout *rel-123*
Make sure you're on your release branch
git fetch *bob*
Fetch the latest changes from a team member. Repeat for all team members.
git checkout *bob/rel-123*
git rebase *rel-123*
git tag *new-rel-123*
git checkout *rel-123*
git merge --ff-only *new-rel-123*
git tag -d *new-rel-123*
(i) Switch to team member's work. (ii) Rebase their changes onto the end of yours. (iii) Tag this latest position. (iv) Switch to your work. (v) Bring it up to date with the new changes. (vi) Get rid of the tag. Repeat all six steps for every team members' work.
git pull --rebase origin master
Rebase your combined work onto the end of the latest changes from origin/master
git checkout master
Switch to master
git merge --ff-only *rel-123*
Bring it up to date with your team's changes
git push origin
Push this latest master up to the central server.

Now your other team members must forget their branches and pull down this latest version of master:

git checkout master
Switch to master
git pull --ff-only origin
Get the latest master from the origin

Detached HEAD state

The only difference here is that you are not attached to a named branch. You can do all operations normally, but when you switch to another branch you may find it difficult to return here. So before you switch you may want to create a new branch or tag at this point.

Version 1.1 of Travis Perkins' git cheat sheet, by Nik Silver
Based on work by Jan Krüger: <jk@jk.gs>
Which is in turn based on work by Zack Rusin, http://zrusin.blogspot.com/