**NOTE ABOUT FAST-FORWARDS**

When an update changes a branch (or more in general, a ref) that used to point at commit A to point at

another commit B, it is called a fast-forward update if and only if B is a descendant of A.

In a fast-forward update from A to B, the set of commits that the original commit A built on top of is

a subset of the commits the new commit B builds on top of. Hence, it does not lose any history.

In contrast, a non-fast-forward update will lose history. For example, suppose you and somebody else

started at the same commit X, and you built a history leading to commit B while the other person built

a history leading to commit A. The history looks like this:

B

/

---X---A

Further suppose that the other person already pushed changes leading to A back to the original

repository from which you two obtained the original commit X.

The push done by the other person updated the branch that used to point at commit X to point at commit

A. It is a fast-forward.

But if you try to push, you will attempt to update the branch (that now points at A) with commit B.

This does not fast-forward. If you did so, the changes introduced by commit A will be lost, because

everybody will now start building on top of B.

The command by default does not allow an update that is not a fast-forward to prevent such loss of

history.

If you do not want to lose your work (history from X to B) or the work by the other person (history

from X to A), you would need to first fetch the history from the repository, create a history that

contains changes done by both parties, and push the result back.

You can perform "git pull", resolve potential conflicts, and "git push" the result. A "git pull" will

create a merge commit C between commits A and B.

B---C

/ /

---X---A

Updating A with the resulting merge commit will fast-forward and your push will be accepted.

Alternatively, you can rebase your change between X and B on top of A, with "git pull --rebase", and

push the result back. The rebase will create a new commit D that builds the change between X and B on

top of A.

B D

/ /

---X---A

Again, updating A with this commit will fast-forward and your push will be accepted.

There is another common situation where you may encounter non-fast-forward rejection when you try to

push, and it is possible even when you are pushing into a repository nobody else pushes into. After you

push commit A yourself (in the first picture in this section), replace it with "git commit --amend" to

produce commit B, and you try to push it out, because forgot that you have pushed A out already. In

such a case, and only if you are certain that nobody in the meantime fetched your earlier commit A (and

started building on top of it), you can run "git push --force" to overwrite it. In other words, "git

push --force" is a method reserved for a case where you do mean to lose history.

**EXAMPLES**

**git push**

Works like git push **<remote>**, where <remote> is the current branch's remote (or origin, if no remote is configured for the current branch).

**git push origin**

Without additional configuration, pushes the current branch to the configured upstream

(remote.origin.merge configuration variable) if it has the same name. as the current branch, and errors out without pushing otherwise.

The default behavior of this command when no <refspec> is given can be configured by setting the push option of the remote, or the **push.default** configuration variable.

For example, to default to pushing only the current branch to origin. use git config **remote.origin.push HEAD**. Any valid <refspec> (like the ones in the examples below) can be configured as the default for git push origin.

**git push origin :**

Push "matching" branches to origin. See <refspec> in the OPTIONS section above for a description of matching" branches.

**git push origin master**

Find a ref that matches master in the source repository (most likely, it would find

refs/heads/master), and update the same ref (e.g. refs/heads/master) in origin repository with it.

If master did not exist remotely, it would be created.

**git push origin HEAD**

A handy way to push the current branch to the same name on the remote.

**git push mothership master:satellite/master dev:satellite/dev**

Use the source ref that matches master (e.g. refs/heads/master) to update the ref that matches

satellite/master (most probably refs/remotes/satellite/master) in the mothership repository; do the

same for dev and satellite/dev.

See the section describing <refspec>... above for a discussion of the matching semantics.

This is to emulate git fetch run on the mothership using git push that is run in the opposite

direction in order to integrate the work done on satellite, and is often necessary when you can

only make connection in one way (i.e. satellite can ssh into mothership but mothership cannot

initiate connection to satellite because the latter is behind a firewall or does not run sshd).

After running this git push on the satellite machine, you would ssh into the mothership and run git

merge there to complete the emulation of git pull that were run on mothership to pull changes made

on satellite.

**git push origin HEAD:master**

Push the current branch to the remote ref matching master in the origin repository. This form is

convenient to push the current branch without thinking about its local name.

**git push origin master:refs/heads/experimental**

Create the branch experimental in the origin repository by copying the current master branch. This

form is only needed to create a new branch or tag in the remote repository when the local name and

the remote name are different; otherwise, the ref name on its own will work.

**git push origin :experimental**

Find a ref that matches experimental in the origin repository (e.g. refs/heads/experimental), and

delete it.

**git push origin +dev:master**

Update the origin repository's master branch with the dev branch, allowing non-fast-forward

updates. This can leave unreferenced commits dangling in the origin repository. Consider the

following situation, where a fast-forward is not possible:

o---o---o---A---B origin/master

\

X---Y---Z dev

The above command would change the origin repository to

A---B (unnamed branch)

/

o---o---o---X---Y---Z master

Commits A and B would no longer belong to a branch with a symbolic name, and so would be

unreachable. As such, these commits would be removed by a git gc command on the origin repository.