98-172 : Great Practical Ideas for Computer Scientists

Making It Never Happen Again

Basic Terminology

A commit is a set of changes to the files tracked by your repository. This can include files added, files removed, and files changed.

To commit is to add a new commit to the repository history.

(Unshared Repository) I would like to...

I would like to **Setup Git**...

To setup git for regular usage, you should set the following configurations:

- `git config --global user.name "<your name here>"`
- `git config --global user.email "<your e-mail address here>"`
- `git config --global color.ui auto`
- `git config --global color.interactive auto`

It also may be helpful to create a file in your repository named `.gitignore`. Put patterns in this file, one per line, to specify files you would like git to ignore. For example, to ignore Vim swap files, add the lines `*.swp` and `*.swo`.

I would like to **Create A New Local Repository**...

To create a new local repository, `cd` into the directory you want to be a git repository and use the command:

```
git init
```

I would like to **Add A File To The Repository**...

To add a file to the respository, use the `add` command:

```
git add <file>
```

You can use "`git add .`" to add everything (although this is ill-advised).

I would like to **Commit A Set Of Changes**...

All Of Them Use the command `git commit -a` to commit all changes.

Some Of Them Use the command `git commit <file>` to commit the changes to a particular file.

I would like to **See My Uncommited Changes**...

Overview To see an overview of your uncommited changes, use the command "`git status`"

Details To see the details of your uncommited changes, use the command "`git diff HEAD`"

I would like to **See All Previous Commits**...

If you want to see a list of all previous commits, then you should use the command "`git log`"

I would like to **Reverse A Commit**...

To reverse a particular commit, first use "`git log`" to find the commit id (which will look something like 89c1b54386bca4e2947d0e61861d318aa0b7556b). Then, use

```
git revert <commit_id>
```

to actually reverse the commit. Git will create *a new commit* that reverses the changes of the one you specified.

I would like to **Discard My Uncommited Changes**...

If you decide that the changes you made since your last commit weren't helpful, and you would like to *get rid of them forever*, then you should use the command:

```
git reset --hard
```

(Shared Repository) We would like to...

All things from the "I would like to" section apply here as well.

We would like to **Create A New Central Repository**...

First, choose a directory for your local version of the shared repository, let's call it `LOCAL_REPO`. Then, to start the setup, run the following commands:

- `cd LOCAL_REPO`
- `git init`
- `touch README` (You can replace `README` with any file that you would like to put in your repository)
- `git add README`
- `git commit -m "Initial Commit"`

Now, we will create the central version of the repository, let's call it `CENTRAL_REPO`. This directory should not be edited directly by any of you (after this section), but it is necessary that both of you have access. If it is on afs, then you should use the command "`fs la . <andrewid> rlidwk`" to give your group members access to it.

- `cd CENTRAL_REPO`
- `git init --bare`

Finally, go back to your local version, and set it up to work with the central one:

- `cd LOCAL_REPO`
- `git push --set-upstream PATH_TO_CENTRAL_REPO master`

We would like to **Get Our Own Copies of the Repository**...

Assuming there is already a "central repository", your group members should get a copy of it (you already have one from the previous step) by running the command:

```
git clone <path_to_central_repository>
```

We would like to **Send Our Changes To Everyone**...

If you are happy with the commits you have made, and you want to send them to the main repository so that everyone can see them, you should use the command

```
git push
```

We would like to **Receive Changes From Everyone**...

Similarly, if you would like to take everyone's changes and pull them into your version of the repository, you should use the command

```
git pull
```

git may tell you that there is a merge conflict when you do this, because it is possible that you and someone else both changed something. In that case, you should...

We would like to **Resolve A Merge Conflict**...

Open the file or files that git says it cannot automatically merge. There will be lines that start with ">>>>>>>" and "<<<<<<<", the lines between these are the differences that git could not figure out how to merge by itself. If you delete these lines and make the files look "correct", then use the command "`git merge`" to finalize the merge. You will be asked to confirm the merge commit message. Finally, push the merge to the central repository.

The Staging Area

So far, all of the commands discussed have dealt with the files in your directory (the "working tree"), the local repository, and possibly a remote repository. There is actually a fourth area where your changes can be stored, called the "index", which is logically located between the working tree and the repository. The index is a sort of staging area for commits, giving you the ability to control and inspect exactly what will be included in a commit. When something is in the index, it is said to be "staged".

To "stage" changes (to add changes from your working tree to the index), you can use various forms of the `git add` command.

- `git add <files>` - add the listed files to the index
- `git add -p` - interactively add "hunks" to the index. This examines all files tracked by the repository, detects the sections that have individually.
- `git add -p <files>` - like `git add -p`, but only looks at the listed files.

You can examine the staged changes with `git diff --cached`.

If you have staged changes which you wish to remove from the index, the command `git reset` can remove them:

- `git reset <files>` - unstage all changes from the listed files
- `git reset -p` - interactively remove staged hunks from the index
- `git reset -p <files>` - probably what you expect
- `git reset --mixed` - unstage all changes in the index

Once you have modified the index to your liking, you can commit the staged changes with a simple `git commit` (no arguments!).

Working On A Large Feature

If you'd like to make a lot of commits and keep them logically seperate for a while, while still sharing them with other people, you can use branches. This is frequently used when adding a large feature to a project: you work on the new feature in a new branch, leaving the master branch clean for others (or yourself) to make bugfixes; then, when the new feature is stable, you merge it back into the master branch.

To create a new local branch, you can say `git checkout -b <branchname>`. (Yes, the naming of this command is kind of inane. It's equivalent to running `git branch <branchname>` to create the branch, and then `git checkout branchname` to switch to it.)

You can see what branches exist using `git branch` with no arguments. The one you are currently on will have a star next to it.

Now you can make changes and commit as usual on the new branch. You can switch between branches using `git checkout <branchname>`.

To share your branch with everyone else, do `git push <branchname>`. You'll also want to run `git branch --set-upstream <branchname> remotes/origin/<branchname>` so that when you pull, git will know how to update your new branch.

To use a branch that someone else has added to a remote repository, do `git checkout --track -b <branchname> origin/<branchname>`.

When you're done with the new branch and ready to combine it into your main one, switch to the master branch (using `git checkout master`) and do `git merge <branchname>`. This will pull in all the changes from the branch (as usual when merging, you may have to resolve conflicting changes by hand).