GAP on GitHub: Quickstart

Configuration

You only need to do this once:

```
git config --global user.name "Your Name"
git config --global user.email you@yourdomain.com
```

This data ends up in commits, so do it now before you forget!

Get the GAP source code

```
git clone git@github.com:gap-system/gap.git
```

Branch often

A new branch is like an independent copy of the source code.
To switch to a new branch before editing, call:

```
git checkout master          switch to the starting point
git branch new_branch_name   create new branch
git checkout new_branch_name switch to new branch
```

Without an argument, the list of branches is displayed:

```
git branch
master
* new_branch_name
```

where the star marks the current branch.
When you finished, delete unused branches:

```
git branch -d branch_to_delete
```

Who did what?

Each change recorded by git is called a commit.
To examine commit history, call

```
git show  show the most recent commit (ie current head)
git log   commit list in reverse chronological order
```

What have I done?

Probably the most important command. Example output:

```
git status
  On branch new_branch_name        // current branch name
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: modified_file.g  // file you just edited

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

        new_file.g                 // file you just added

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

Preparing your Commit

When ready, tell git which changes do you want to commit:

```
git add filename          add particular file
git add .                 add all modified and new
```

The status command then lists the staged changes:

```
git status
On branch new_branch_name
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

      modified: modified_file.g
      new file: new_file.g
```

Commit!

The commit command will permanently record the staged changes.
The new commit becomes the new branch head. To write commit message in an editor, call

```
git commit
```

otherwise just call

```
git commit -m "My Commit Message"
```

Commits cannot be changed, but they can be discarded and re-done with the `--amend` switch. **Never amend commits that you have already shared with somebody.**

Summary

Workspace is the file system: files that you can edit

```
git add filename          copy file to staging
git reset HEAD filename   copy staged file back
```

Staging is a special area inside the git repository

```
git commit                commit all staged files
```

Commits are the permanently recorded history

```
git checkout -- filename  copy filename from
                          repository to workspace
```

Keeping in sync

If you have write access to the GAP repository, next time before you start any changes you should incorporate changes from the master branch of the remote repository with

```
git pull origin master          fetch and merge
```

or (to linearise history if you have not pushed changes) with

```
git pull --rebase origin master fetch and rebase
```

Resolve conflicts, if there are any (see below), then switch to the top of the branch:

```
git checkout master       switch to the starting point
```

If you don't have write access, see `https://help.github.com/articles/syncing-a-fork/` on syncing a fork.

Merging

A commit with more than one parent is a merge commit:

```
git merge other_branch
```

incorporates other branch/commit. If there is no conflict, this automatically creates a new merge commit. Otherwise, the conflicting regions are marked like this:

```
Lines that are either unchanged from a common ancestor,
or cleanly resolved because only one side changed.
<<<<<<< yours:source_file.g
It's frustrating to resolve conflicts.
Let's have coffee.
=======
It's easy to resolve conflicts with Git.
>>>>>>> theirs:source_file.g
Another line that is cleanly resolved or unmodified.
```

Edit as needed; To finish, run one of:

```
git commit        commit your merge conflict resolution
git merge --abort discard merge attempt
```

Branch Heads

A git branch is just a pointer to a commit. This commit is called the branch `HEAD`. You can point it elsewhere with (`--hard`) or without (`--soft`, less common) resetting the actual files. That is, the following discards content of the current branch and makes it indistinguishable from a new branch that started at `new_head_commit`:

```
git reset --hard new_head_commit
```

There are various ways to specify a commit to reset to:

```
994bd8d4b279c549cb01ef4e684c246e555f6bae  40-digit sha1
994bd8d                  the first few digits of the sha1
branch_name      the name of another branch pointing to it
4.8.beta0                a tag in the GAP git repository
origin/master    the master branch in the remote origin
```

Most important git commands!

Git comes with extensive documentation:

```
git help -i   most commonly used Git commands
git help -a                list of all Git commands
git help -g                list of available Git guides
```

Adapted from http://github.com/sagemath/git-trac-command/raw/master/doc/git-cheat-sheet.pdf